NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

19980102 154

DESIGN, CONSTRUCTION AND PROGRAMMING
OF A MICROCONTROLLER-BASED TESTBENCH
SUITABLE FOR RADIATION TESTING OF
MICROELECTRONIC CIRCUITS

by

John A. Thompson

March 1997

Thesis Advisor: Douglas J. Fouts

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1997	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE DESIGN, CONSTRUCTION AND PROGRAMMING OF A MICROCONTROLLER-BASED TESTBENCH SUITABLE FOR RADIATION TESTING OF MICROELECTRONIC CIRCUITS	5. FUNDING NUMBERS
6. AUTHOR(S) Thompson, John A.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

This thesis describes the design, construction, and programming of a microcontroller-based testbench suitable for radiation testing microelectronic integrated circuits. It will be used to test circuits fabricated using the Low Temperature Gallium Arsenide (LT GaAs) fabrication process developed by the Naval Postgraduate School and the Naval Research Laboratory. The testbench will be used to test for sensitivity to Single Event Upsets (changes in logic level due to impact by high energy ions). Due to the spurious radiation around the particle accelerator, it will be remotely operated via a serial communication port. Radiation hardened components will eventually be used throughout, although for cost-savings, non-radiation hardened components are used in the initial design described here. The test bench is built around the Intel 87C51 four-port microcontroller. As part of this research, it will be programmed to test two memory chips, one manufactured by Motorola Inc. and one by Vitesse Semiconductor Corporation. The Motorola chip requires that a special chip carrier with logic translation and output drivers be designed prior to testing.

14. SUBJECT TERMS Testbench, Radiation Testing, Memory Testing, Micrporocessor-Controlled Test Equipment	15. NO. OF PAGES 130
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

DESIGN, CONSTRUCTION AND PROGRAMMING OF A MICROCONTROLLER-BASED TESTBENCH SUITABLE FOR RADIATION TESTING OF MICROELECTRONIC CIRCUITS

John A. Thompson
Lieutenant, United States Coast Guard
B.S., Old Dominion University, 1992

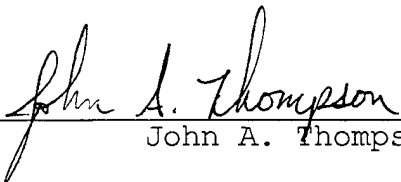Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING
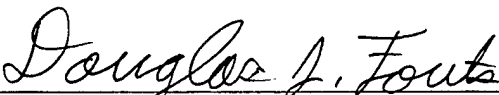
from the

NAVAL POSTGRADUATE SCHOOL
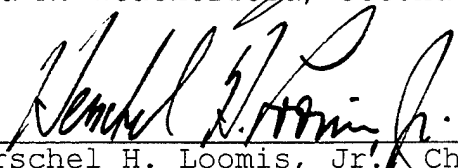**March 1997**

Author: _____

John A. Thompson

Approved by: _____

Douglas J. Fouts, Thesis Advisor

_____

Todd R. Weatherford, Second Reader

_____

Herschel H. Loomis, Jr., Chairman
Department of Electrical and Computer Engineering

iii

ABSTRACT

This thesis describes the design, construction, and programming of a microcontroller-based testbench suitable for radiation testing microelectronic integrated circuits. It will be used to test circuits fabricated using the Low Temperature Gallium Arsenide (LT GaAs) fabrication process developed by the Naval Postgraduate School and the Naval Research Laboratory. The testbench will be used to test for sensitivity to Single Event Upsets (changes in logic level due to impact by high energy ions). Due to the spurious radiation around the particle accelerator, it will be remotely operated via a serial communication port. Radiation hardened components will eventually be used throughout, although for cost-savings, non-radiation hardened components are used in the initial design described here. The test bench is built around the Intel 87C51 four-port microcontroller. As part of this research, it will be programmed to test two memory chips, one manufactured by Motorola Inc. and one by Vitesse Semiconductor Corporation. The Motorola chip requires that a special chip carrier with logic translation and output drivers be designed prior to testing.

TABLE OF CONTENTS

I. INTRODUCTION

The research project described here is but a small part of a much larger and potentially very beneficial project at the Naval Postgraduate School (NPS). Researchers at NPS, in conjunction with researchers at the Naval Research Laboratory and two companies, have been developing a new technique for fabricating gallium arsenide (GaAs) digital integrated circuits with the intent to make them immune to soft errors. Soft errors (or Single Event Upsets, SEUs) are caused by high energy particles which strike the memory or logic cell and impart energy into it. This energy can be enough to switch logic levels, thereby corrupting the stored data. A satellite in geosynchronous orbit using present gallium arsenide circuits can expect from 1 to 1000 errors per day per Mbit of stored information (Weatherford, et al, 1996).

The new process involves growing an additional epitaxial layer of GaAs on the surface of the wafer prior to growing the layers where the transistors are to be fabricated. This additional layer is grown at a much lower temperature than normal (220° vice 600°) which dramatically shortens the carrier lifetime. The higher recombination rate neutralizes the charge caused by the particle before the charge can change the stored logic level. Once perfected, the LT GaAs process can potentially reduce the soft error rate by 8 orders of magnitude (Weatherford, et al, 1996). This new process will require no changes to existing circuit designs, only an additional step in the wafer growth prior to integrated circuit fabrication.

This particular research project involves designing, fabricating, and programming a microcontroller-based testbench for testing the circuits being fabricated using the LT GaAs

1

process. The testbench must be able to test a variety of electronic circuits. Therefore, it must be easily reprogrammable and all the control signals must be made available to the circuit under test. Each circuit will be bench-tested for functionality, then tested for sensitivity to SEUs. A particle accelerator will be used as the source of radiation for the SEU tests. Due to the high radiation levels around the accelerator, the testbench will require radiation-hardened components. Non-radiation hardened components will be used during initial development and replaced once the correct operation of the testbench verified. This will prevent the destruction of the expensive radiation-hardened circuitry during troubleshooting. A remote communication link will also be included so that operators will not have to enter the accelerator room during tests. The testing program being run can be monitored and new programs can be down-loaded via this link. Initial tests included as part of this project will be on two memory chips. Motorola is supplying a 256 X 16-bit SRAM chip in both GaAs and LT GaAs versions. Vitesse is also providing a 256 X 4 bit SRAM memory chip.

This thesis is also intended to be a technical manual for the testbench itself. In addition, the testing algorithms and implementation details will be provided for the two memory chips. Chapter II provides a basic operational description of the testbench. Detailed theory of operation is found in Chapter III. Chapter IV and Chapter V are excerpts from the data sheets for the i87C51 microcontroller published by Philips Semiconductors. The architectural description of the microcontroller is in Chapter IV and the instruction set is presented in Chapter V. Chapter VI covers the operating system, PAULMON, written by Mr. Paul Stoffregen of Oregon State University. Details on the two SRAM chips from Motorola

and Vitesse and the testing algorithms are located in Chapter VII. Included are the pin outs, the flow charts and details of the hardware adapters required in order to interface the chip with the testbench. Chapter VIII wraps up this project and provides recommendations for future work. The appendices include all the schematic drawings, PC Board layout drawings and assembly language code as well as a detailed description on how to build a printed circuit board using the equipment at NPS.

II. SYSTEM LEVEL DESIGN

A. OVERVIEW

The testbench is built around the Intel 87C51 microcontroller. This chip is compatible with the radiation-hardened UT69RH051 from United Technologies which will later be substituted. Radiation hardened chips are extremely expensive, sometimes costing hundreds of times as much as their non-radiation hardened counterparts.

Six ports are provided on the testbench to connect to the unit-under-test (UUT). Two of these interface directly to the microcontroller, CONN 0 and CONN 1. These ports provide access to the address bus, data bus, and various control signals. Three of the ports access the microcontroller via a 32-bit to 8-bit interface. The testbench will be connected to a remote terminal via an RS-232 port (CONN 0) to provide an operator interface as well as to keep the operator well clear of any spurious radiation around the particle accelerator. The testbench itself will be shielded with lead bricks while performing radiation testing. This may not be necessary once radiation hardened components are substituted.

B. MICROCONTROLLER AND MEMORY SUBSYSTEM

The microcontroller used in the testbench has four programmable parallel input/output ports. Two of these are used to provide a 16-bit address bus. The lower byte of the address must be latched because this port is also used as the system data bus. A second 8-bit data bus is also available. The remaining port is unused as a parallel port. Instead, its pins provide a serial communication port which is used in the test bench to provide remote communications while in a radioactive environment. The microprocessor runs at 7.3728

5

MHz, which can be divided evenly to provide the desired baud
rate for the serial communication port. A basic block diagram
is shown in Figure 2.1 below.
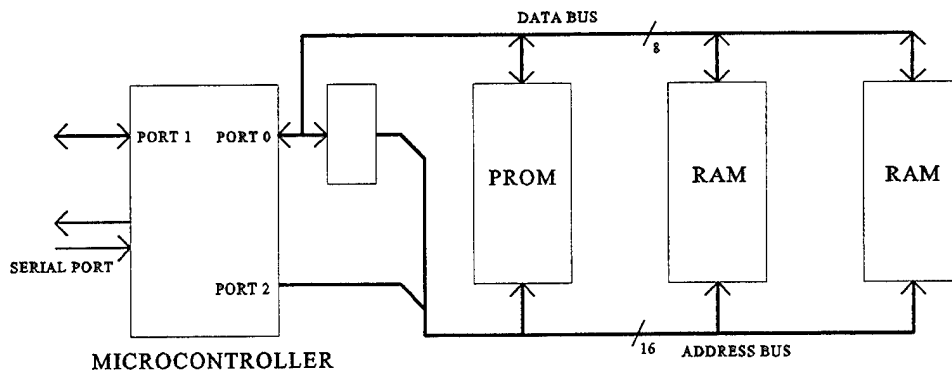
Figure 2.1   Testbench Microcontroller and Memory

     The microprocessor has the capacity for dual 64 Kbyte
address spaces, one for data and one for program code. In the
testbench, these address spaces have been combined into a
single 64 Kbyte address space. If internal PROM is present in
the microcontroller, it is recommended that it not be enabled.
It can be enabled, however, by switching a jumper at the
expense of the lower half of the external PROM. Internal
PROM, if present, resides in the lowest 4 Kbytes of the
address space. The external PROM normally resides in the
lowest 8 Kbytes of the memory map.

     Two RAM chips are present in the testbench. Together,
they occupy the next two 8 Kbytes above the PROM in the memory
map. The upper 32 Kbytes of memory space is available to the
unit under test. It is possible to add additional RAM
externally using some of this space, while reserving part for
the unit being tested. The address space is shown graphically
in Figure 2.2.

0000h	2000h	4000h	6000h	8000h	A000h	C000h	E000h	FFFFh
PROM	RAM	RAM	CONT. SPACE	TEST SPACE	TEST SPACE	TEST SPACE	TEST SPACE	

Figure 2.2   Address Space

A more detailed description of the microcontroller and memory subsystem is located in Section 3.A.   Complete schematic diagrams are located in Appendix B.

C.   PERIPHERAL INTERFACE SUBSYSTEM

The testbench has six ports (CONN 0 through CONN 5) for communication with the outside world, as shown in Figure 2.3. CONN 0 is the serial communication port previously described. CONN 1 is the data buses (port 0 and port 1) and CONN 2 is the address bus, which are directly connected to the microcontroller.   It is anticipated that this testbench may be used to test electronic memories that are much larger than the capacity of the testbench itself (eight bits).   Therefore, interface circuitry is provided to combine four bytes of data into a single 32-bit word and vice-versa.   This capability is replicated so that a 32-bit address word and a 32-bit data word can be created byte by byte.   CONN 3 is intended to be used as a 32-bit address bus.   CONN 4 is intended to be a 32-bit data bus.   Each of these bidirectional ports can be individually byte accessed to allow maximum flexibility.   In addition, there is a 32-bit port, CONN 5, that is restricted to input data only.   Incoming data is latched and passed to the data bus over four read cycles.   CONN 5 can be used as a 32-bit data bus coming from the unit-under-test, while CONN 4 can be used as a 32-bit output bus going to the unit being tested.
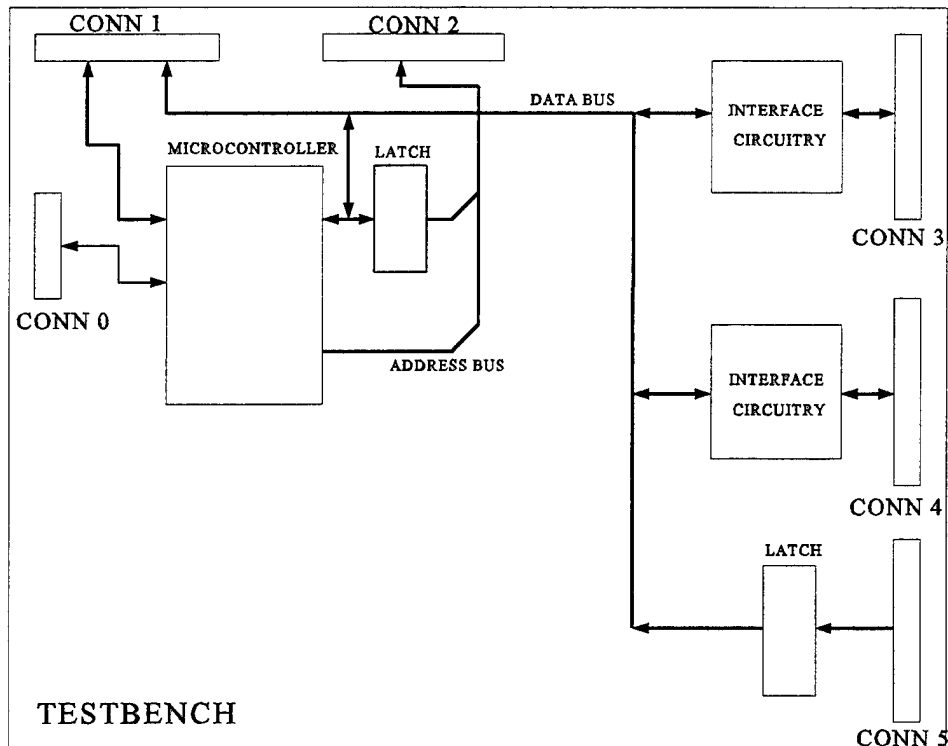
Figure 2.3 Testbench Ports

The fourth 8 Kbyte page (just above the RAMs) in the memory map is dedicated to control functions. These functions select the peripheral interface, the individual port to be accessed and whether it is to be an input or an output.

A more detailed description of the peripheral interface subsystem is located in Section 3.B. Complete schematic diagrams are located in Appendix B.

D. POWER SUPPLY

The power supply used in the testbench is a commercially available unit manufactured by ACME Electric Corporation. It actually has three outputs (+5 VDC, +12 VDC, -12 VDC), although only the +5 VDC output is used internally. All three

outputs are available externally via the three terminals on the rear of the testbench. Separate fuses are provided for each terminal as well as for the internal +5 VDC supply. The power supply is equipped with over-voltage and over-current protection on the +5 VDC output only. A more detailed description is located in Section 3.C. The schematic diagram is located in Appendix B.

III. DESIGN DETAILS AND THEORY OF OPERATION

A. MICROCONTROLLER AND MEMORY SUBSYSTEM

The microcontroller, designated U1, used in the testbench is either the UT69RH051 radiation-hardened model manufactured by United Technologies or the non-radiation-hardened equivalent, the i87C51 manufactured by Intel. Pin-for-pin, both units are essentially identical with the largest difference being the lack of internal PROM in the UT69RH051. Due to this difference, the testbench is designed not to use internal PROM at all no matter which chip is used. This feature can be overcome by switching a jumper if the i87C51 is used. Other differences include the lack of an idle mode, a power-down mode, and an on-circuit emulation mode in the UT69RH051. The UT69RH051 microcontroller requires a longer reset time (24 clock periods) than the i87C51. It is a more rugged device than the i87C51 and can withstand a much wider temperature range.

Both microcontrollers have a separate address space for program and data memory. Up to 64 Kbytes can be addressed for each. However, in the testbench, both address spaces have been combined into a single 64 Kbyte space. This space is divided into eight 8 Kbyte pages. The external PROM resides in the lowest 8 Kbyte page. If the internal PROM of the i87C51 is enabled, it occupies the lowest 8 to 32 Kbytes, depending on which version is used. To enable the internal PROM, $\overline{EA}$ (pin 31) must be tied high via jumper J2. *NOTICE: Jumpers J1 and J2 must never be in place at the same time. This will short out the power supply.*

The microcontroller features four 8-bit bidirectional parallel ports of which three are used in the testbench. Port 0 and Port 2 together are used to form a 16-bit address. The

11

16-bit address is directly available to the external world via connector CONN 2. Port 0 is latched by an octal latch (U5) so that it may also be used as an eight-bit data bus. This is the primary data bus for the test bench. Port 1 is used as a second eight-bit data bus or, together with port 0, as a 16-bit data bus. Both Port 0 and Port 1 are available to the external world via connector CONN 1.

Port 3 is not used in the testbench, though many of its pins are used for their secondary function. Pins 10 (*RXD*) and 11 (*TXD*) provide a full-duplex programmable serial port. This port features a receive buffer, framing error detection and automatic address recognition. The TTL logic levels available on these two pins are converted to RS-232 logic level in the MAX233 (U6). This device makes use of charge-pumping to convert the 0 to +5 VDC TTL levels to the -10 VDC to +10 VDC RS-232 levels while using a single +5 VDC supply. The serial port is available to the external world via connector CONN 0.

Pin 16 is the $\overline{WRITE}$ signal and pin 17 is the $\overline{READ}$ signal. These are routed throughout the testbench and are made available to the external world via CONN 2 with the address bus. Pin 9 is the *RESET* signal. This pin must be held high for 24 clock cycles in order to guarantee that all the registers in the microcontroller are cleared. A resistor-capacitor series network provides this function on power-up. A front panel push-button switch is also available to reset the testbench. The *RESET* signal also sets the peripheral interface devices and is available to the external world via connector CONN 2 with the address bus.

Pins 18 and 19 are connected to the crystal. Pin 18 is the input to the internal oscillator of the microcontroller and pin 19 is the output. The crystal frequency is 7.3728 MHz. This frequency can be evenly divided down to provide the

12

common serial communication baud rates (e.g. 2400 baud, 4800 baud, 9600 baud, etc).

The address space is divided into 8 Kbyte pages as shown in Figure 3.1 by a 3-to-8 decoder (U8). Address lines A13 through A15 are used to select each page. Four of these page selects (or chip selects) serve no internal purpose to the testbench and are available to the external world via connector CONN 2 with the address bus. The four lower pages of the address space are internal to the testbench. Page 1 is dedicated to the PROM, U2. The radiation-hardened version of this chip is the UT28F64. The non-radiation-hardened version is the AM2764A (EPROM) or the AT28C64 (EEPROM). The system RAM chips reside in pages 2 and 3. These are UT67164
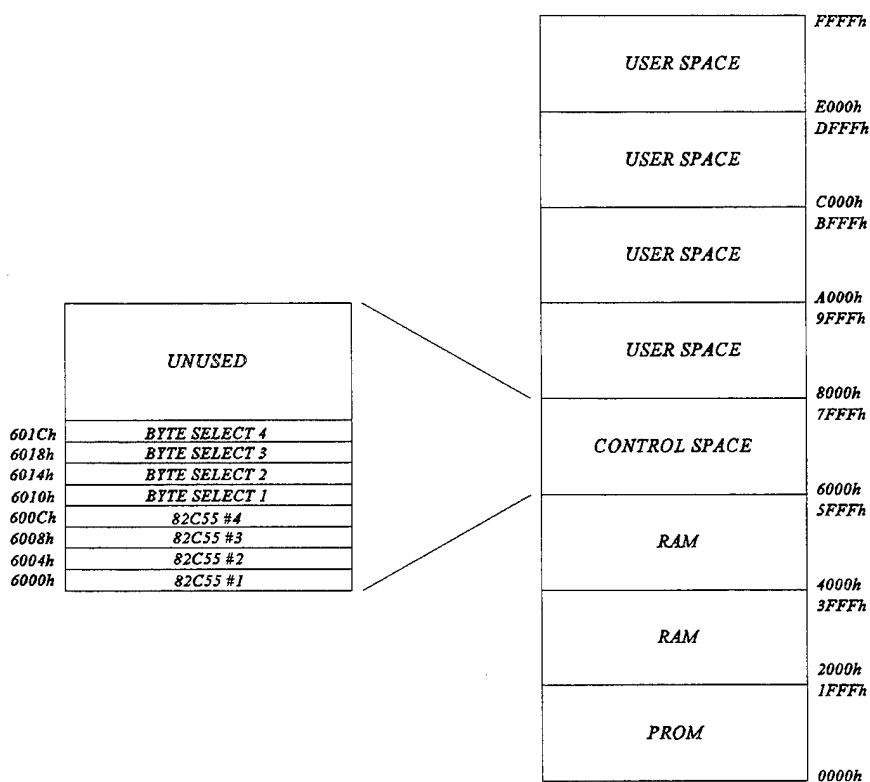
Figure 3.1 Testbench Memory Map

13

radiation-hardened chips or 61C64 chips for non-radiation-hardened applications. The pin-out for the RAM chips is exactly the same as for the PROM chip with the exception of pins 26 and 27. These are the $\overline{WRITE}$ lines for the RAMs and $\overline{PE}$ (program Enable) for the PROM.

The fourth 8 Kbyte page in the address space is used for control purposes. A second 3-to-8 decoder (U9) further divides this page up to provide chip select signals for each of the peripheral interface devices and four byte-select signals. These byte-select signals select which byte of the 32-bit input data bus (CONN 5) will be connected to the internal data bus of the testbench. By activating each of these byte-select lines in sequence, a 32-bit data word can be read into the microcontroller one byte at a time.

A detailed schematic of the microcontroller and memory subsystem is located in Appendix B.

B.   PERIPHERAL INTERFACE SUBSYSTEM

The peripheral interface subsystem includes all the hardware necessary to mate the 32-bit ports (CONN 3, CONN 4 and CONN 5) to the eight-bit testbench. The hardware associated with CONN 3 and CONN 4 is identical. Two programmable peripheral interface devices (HS-82C55ARH or 82C55A), each providing two eight-bit ports, provide the 32 pins needed. Actually, the interface devices (U9 through U12) have three ports apiece, however one of the ports (Port C) is used for control purposes. Port A and Port B of each interface device goes through an octal bus transceiver. The transceiver prevents the 32-bit data from showing up on the associated connector until all four bytes are written and the correct control signal comes from Port C.

The lower nibble of Port C contains the control signals

14

which determine the direction of data flow and also enable the transceiver. The high nibble is unused. When the transceiver is disabled, the buses on either side are effectively disconnected. Once the four bytes of data are written to Ports A and B of the two interface devices, $XXXX0011_b$ (X represents don't cares) must be written to Port C of both interface devices to connect the four bytes to the associated connector. In order to receive 32-bits of data, $XXXX0000_b$ must be written to Port C of both interface devices, then the bytes must be read sequentially. The 32-bit input data word on the connector must be held for at least five read cycles.

The individual interface devices are selected by U9, as previously discussed. Once selected, the lowest two bits of the address bus determine which port is to be selected. Other inputs to the interface devices include the $\overline{READ}$ and $\overline{WRITE}$ signals which determine if data is being written to the port or read from it. Of course, the data bus is also an input to the four peripheral interface devices.

CONN 5 is unique in that it is restricted to data-input functions only. The four bytes of data are latched simultaneously into four octal D flip-flops (U21 through U24) and read a byte at a time over four read cycles. This port has the advantage over CONN 4 in that the data only has to be held stable for one read cycle. The four bytes are latched together by the *DATA LATCH* signal. This signal is generated by the logical AND of A15 and the $\overline{READ}$ signal. The octal flip-flops enable their outputs when a signal comes from U9 ($\overline{CS\,BYTE\,1}$ to $\overline{CS\,BYTE4}$), as previously discussed.

Detailed schematics can be found in Appendix B.

C. POWER SUPPLY

The 1A2 power supply module is a commercially available

product manufactured by ACME Electric Corporation. It is a low profile, unenclosed, three-output switching power supply, model number LSWT-3031. Mounted inside and to the rear of the top cover, it provides the +5 VDC supply to the testbench and also provides three voltage sources (+5 VDC, +12 VDC, -12 VDC) to the accessory plugs on the back of the testbench. All sources are fused so that an over-current condition at the accessory plugs will not affect the supply to the testbench. This model is equipped with both over-voltage and over-current protection on the +5 VDC output. Table 3.1 lists the specifications of the LSWT-3031 as listed on the data sheets and is reproduced here with permission of ACME Electric Corporation. Table 3.2 and Figure 3.2 are also from ACME's data sheets for the LSWT-3031 and show the input and output pin connections.

D. CASE, PC BOARD, AND CABLING DETAILS

The case is a two-part aluminum enclosure manufactured by Precision Fabrication Technologies (PFT) Inc. The power supply module (1A2) is fastened to the back of the top part. Also attached to the top are all the ports, fuses, switches, and accessory power plugs. The lone fuse on the top left of the case is for the +5VDC supply to the testbench itself. The fuses on the rear are for the accessory plugs. The PC board is mounted on nylon stand-offs on the base and connected to the rest of the unit via ribbon cables.

The 8" x 9" printed circuit board (1A1) was fabricated at the Naval Postgraduate School. It is double-clad with nickel-plated copper which forms a power plane on one side and a ground plane on the other. The circuit was laid out using three programs: Easytrax, PCGerber, and Protoboard. Appendix E has explicit details on this process. The outline of the

AC INPUT	
Input Voltage	85-264 Vac
Frequency	47-440 Hz
Current (115 Vac/230 Vac)	0.8A / 0.5A
Inrush Current	15A @ 115Vac, 30A @ 230 Vac
Leakage Current	to EN60950
DC Output	
Output 1	+5 VDC @ 3 Amps
Output 2	+12 VDC @ 1.5 Amps
Output 3	-12 VDC @ 0.2 Amps
Output range	±5% on +5 VDC Output
Line Regulation	±0.5%
Load Regulation	1% Single Output, 3% main Output (50 to 100% load change)
Cross Regulation	±5% typical
Ripple and Noise	1% peak-to-peak typical
Overload Protection	Overload and Short Circuit Protected
Overvoltage Protection	5.6 - 6.6V on +5 VDC Output
Minimum Load	Approximately 10% on +5 VDC Output
Efficiency	65% min at full load
Holdup Time	12 msec
General	
Operating Temperature	0° to 50° C
MTBF	100,000 Hours Minimum
RFI Performance	VDE 0871AA

Table 3.1   Power Supply Specifications

PIN	USE
P1-N	AC Input - Neutral
P1-L	AC Input - Line
P2-1	DC Output - +5 VDC
P2-2	DC Output - +5 VDC
P2-3	DC Output - Return
P2-4	DC Output - Return
P2-5	DC Output - +12 VDC
P2-6	DC Output - -12 VDC

Table 3.2   Pin Assignments for the LSWT-3031 Power Supply
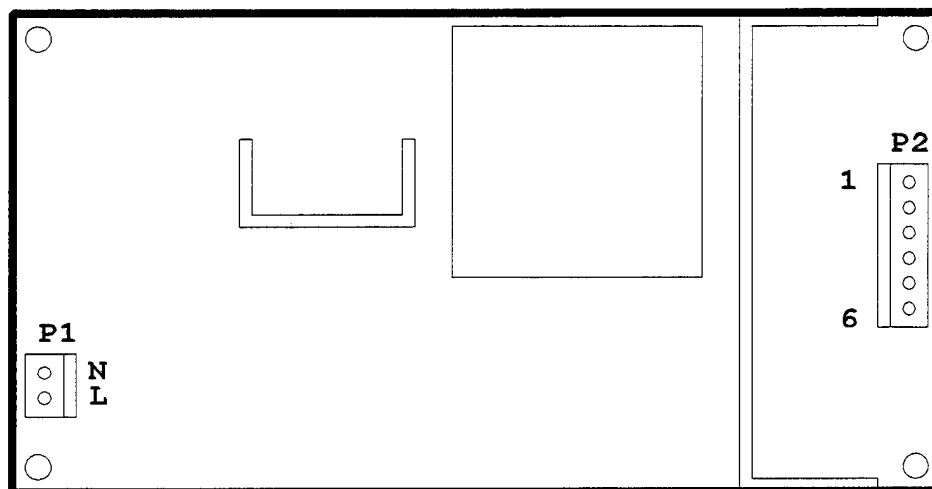
Figure 3.2   LSWT-3031 Pin Locations

traces, pads and vias was milled into the board, isolating them from the power and ground plane.   The ICs are mounted on low-profile sockets for easy replacement.

There are six connectors on the front of the testbench labeled CONN 0 through CONN 5.   CONN 0 is a subminiature D connector that is RS-232 compatible for serial communications directly to the microcontroller.   This is used for communications with the user via a portable PC configured as

18

a terminal. Half the conductors in the ribbon cable connectors (CONN 1 through CONN 5) consist of power and ground. Each conductor that carries a signal is shielded by V_{DD} on one side and ground on the other. This is done in an effort to minimize noise and reduce the effects of the radioactive environment. CONN 1 provides direct access to Port 0 and Port 1 of the microcontroller. This is intended to be either two eight-bit data buses or a single 16-bit data bus. CONN 2 provides direct access to the 16-bit address bus of the microcontroller. Also included on this connector are important control signals such as $\overline{READ}$, $\overline{WRITE}$, RESET, and the page selectors. CONN 3 is intended to be used as a 32-bit address port. This port is indirectly linked to the microcontroller via two peripheral interfaces. It can actually be used as a 32-bit I/O port, two 16-bit I/O ports, or four eight bit I/O ports. If used as a 32-bit address bus, the address must be written via the data bus over four write cycles. Therefore, it is much slower than the 16-bit address bus port. Similarly, CONN 4 is intended to be used as a 32-bit data bus but can also be used as an I/O port, much like CONN 3. CONN 5 can only be used as an input port. It is intended to be used as a 32-bit data input bus but is also byte-accessible. The pin assignments for the connectors follow.

PIN	USE
2	XMT DATA
3	RCV DATA
7	GROUND

Table 3.3   CONN 0 Pin Assignments

19

PIN	USE	PIN	USE	PIN	USE	PIN	USE
1	Vdd	10	P0-4	19	GND	28	P1-5
2	P0-0	11	GND	20	P1-1	29	Vdd
3	GND	12	P0-5	21	Vdd	30	P1-6
4	P0-1	13	Vdd	22	P1-2	31	GND
5	Vdd	14	P0-6	23	GND	32	P1-7
6	P0-2	15	GND	24	P1-3	33	Vdd
7	GND	16	P0-7	25	Vdd	34	N/C
8	P0-3	17	Vdd	26	P1-4		
9	Vdd	18	P1-1	27	GND		

Table 3.4   CONN 1 Pin Assignments

PIN	USE	PIN	USE	PIN	USE	PIN	USE
1	Vdd	14	A6	27	GND	40	*CSC**
2	A0	15	GND	28	A13	41	Vdd
3	GND	15	A7	29	Vdd	42	*CSE**
4	A1	17	Vdd	30	A14	43	GND
5	Vdd	18	A8	31	GND	44	*READ**
6	A2	19	GND	32	A15	45	Vdd
7	GND	20	A9	33	Vdd	46	*WRITE**
8	A3	21	Vdd	34	N/C	47	GND
9	Vdd	22	A10	35	GND	48	*RESET*
10	A4	23	GND	36	*CS8**	49	Vdd
11	GND	24	A11	37	Vdd	50	N/C
12	A5	25	Vdd	38	*CSA**		
13	Vdd	26	A12	39	GND		

Table 3.5   CONN 2 Pin Assignments

PIN	PORT	USE	PIN	PORT	USE	PIN	PORT	USE
1	Vdd		23	GND		45	Vdd	
2	U9PA0	A0	24	U9PB3	A11	46	U10PA6	A22
3	GND		25	Vdd		47	GND	
4	U9PA1	A1	26	U9PB4	A12	48	U10PA7	A23
5	Vdd		27	GND		49	Vdd	
6	U9PA2	A2	28	U9PB5	A13	50	U10PB0	A24
7	GND		29	Vdd		51	GND	
8	U9PA3	A3	30	U9PB6	A14	52	U10PB1	A25
9	Vdd		31	GND		53	Vdd	
10	U9PA4	A4	32	U9PB7	A15	54	U10PB2	A26
11	GND		33	Vdd		55	GND	
12	U9PA5	A5	34	U10PA0	A16	56	U10PB3	A27
13	Vdd		35	GND		57	Vdd	
14	U9PA6	A6	36	U10PA1	A17	58	U10PB4	A28
15	GND		37	Vdd		59	GND	
16	U9PA7	A7	38	U10PA2	A18	60	U10PB5	A29
17	Vdd		39	GND		61	Vdd	
18	U9PB0	A8	40	U10PA3	A19	62	U10PB6	A30
19	GND		41	Vdd		63	GND	
20	U9PB1	A9	42	U10PA4	A20	64	U10PB7	A31
21	Vdd		43	GND				
22	U9PB2	A10	44	U10PA5	A21			

Table 3.6   CONN 3 Pin Assignments

21

PIN	PORT	USE	PIN	PORT	USE	PIN	PORT	USE
1	Vdd		23	GND		45	Vdd	
2	U11PA0	D0	24	U11PB3	D11	46	U12PA6	D22
3	GND		25	Vdd		47	GND	
4	U11PA1	D1	26	U11PB4	D12	48	U12PA7	D23
5	Vdd		27	GND		49	Vdd	
6	U11PA2	D2	28	U11PB5	D13	50	U12PB0	D24
7	GND		29	Vdd		51	GND	
8	U11PA3	D3	30	U11PB6	D14	52	U12PB1	D25
9	Vdd		31	GND		53	Vdd	
10	U11PA4	D4	32	U11PB7	D15	54	U12PB2	D26
11	GND		33	Vdd		55	GND	
12	U11PA5	D5	34	U12PA0	D16	56	U12PB3	D27
13	Vdd		35	GND		57	Vdd	
14	U11PA6	D6	36	U12PA1	D17	58	U12PB4	D28
15	GND		37	Vdd		59	GND	
16	U11PA7	D7	38	U12PA2	D18	60	U12PB5	D29
17	Vdd		39	GND		61	Vdd	
18	U11PB0	D8	40	U12PA3	D19	62	U12PB6	D30
19	GND		41	Vdd		63	GND	
20	U11PB1	D9	42	U12PA4	D20	64	U12PB7	D31
21	Vdd		43	GND				
22	U11PB2	D10	44	U12PA5	D21			

Table 3.7 CONN 4 Pin Assignments

PIN	PORT	USE	PIN	PORT	USE	PIN	PORT	USE
1	U21-D0	D0	23	U22-D3	D11	45	U23-D6	D22
2	Vdd		24	GND		46	Vdd	
3	U21-D1	D1	25	U22-D4	D12	47	U23-D7	D23
4	GND		26	Vdd		48	GND	
5	U21-D2	D2	27	U22-D5	D13	49	U24-D0	D24
6	Vdd		28	GND		50	Vdd	
7	U21-D3	D3	29	U22-D6	D14	51	U24-D1	D25
8	GND		30	Vdd		52	GND	
9	U21-D4	D4	31	U22-D7	D15	53	U24-D2	D26
10	Vdd		32	GND		54	Vdd	
11	U21-D5	D5	33	U23-D0	D16	55	U24-D3	D27
12	GND		34	Vdd		56	GND	
13	U21-D6	D6	35	U23-D1	D17	57	U24-D4	D28
14	Vdd		36	GND		58	Vdd	
15	U21-D7	D7	37	U23-D2	D18	59	U24-D5	D29
16	GND		38	Vdd		60	GND	
17	U22-D0	D8	39	U23-D3	D19	61	U24-D6	D30
18	Vdd		40	GND		62	Vdd	
19	U22-D1	D9	41	U23-D4	D20	63	U24-D7	D31
20	GND		42	Vdd		64	GND	
21	U22-D2	D10	43	U23-D5	D21			
22	Vdd		44	GND				

Table 3.8 CONN 5 Pin Assignments

IV.   INTEL MCS-51 FAMILY ARCHITECTURE

The testbench uses the Intel 87C51 microcontroller or the radiation-hardened equivalent from United Technologies, the UT69RH051. Both are derivatives of the 8051 microcontroller. There are hundreds of different versions of this micro-controller on the market. The 87C51 is a CHMOS version which uses EPROM rather than PROM. The UT69RH051 does not include on-board ROM of any type. Only the information that applies directly to the 87C51 is reproduced here. The information in this chapter is paraphrased from the 80C51 family data sheets published by Philips Semiconductors, dated March 1995. This information can also be found in Intel's "Embedded Microcontrollers and Processors, Volume I", dated 1992.

A.   MEMORY ORGANIZATION

The 80C51 family of microcontrollers all have separate address spaces for program and data memory. The MOVC command reads from program memory; MOVX accesses data memory. Each memory block can have up to 64K addresses. External program and data memory can be combined, if desired, by applying the $\overline{READ}$ and $\overline{PSEN}$ signals to the inputs of an AND gate. The output of the AND gate becomes the read strobe for the combined address space.

All Intel 80C51 microcontrollers have 4 Kbytes of internal PROM. In order to disable this memory and use strictly external PROM, the $\overline{EA}$ (External Access) must be tied to V_{ss}. $\overline{PSEN}$ is the read strobe to external memory. If internal memory is used, $\overline{PSEN}$ is not active for addresses within the first 4 Kbytes.

The 80C51 has three 128-byte blocks of internal 8-bit registers. However, all internal data accesses are limited to

an 8-bit address, which implies a maximum of 256 bytes. The way the 80C51 gets around this is to make two of the blocks share the same addresses. The type of instruction differentiates which block is to be accessed. One of the two blocks is accessible by indirect addressing only. The other is directly addressable and it is in this area that all the Special Function Registers (SFR) are located. Figure 4.1 shows the internal data memory. In the lowest 128 bytes of registers there are 4 banks of 8 registers named R0 through R7. Two bits in the Program Status Word (PSW) select which register bank to use. The default reset value of the stack pointer is 07h. Therefore, the stack, which grows upward, starts at 08h. The next 16 bytes above the register banks form a block of bit-addressable memory space. The 80C51 instruction set includes a wide selection of single-bit instructions and the 128 bits in this area can be directly addressed by these instructions.

B. SPECIAL FUNCTION REGISTERS

Figure 4.2 shows the contents and location of the special function registers. The bulk of this 128-byte block is not implemented on chip. Read accesses to these addresses will, in general, return random data and write accesses will have no effect. The shaded areas in the figure are those areas that are not implemented.

1. Accumulator (ACC)

The accumulator register resides at address E0h. The mnemonics for accumulator-specific instructions refer to the accumulator simply as A.
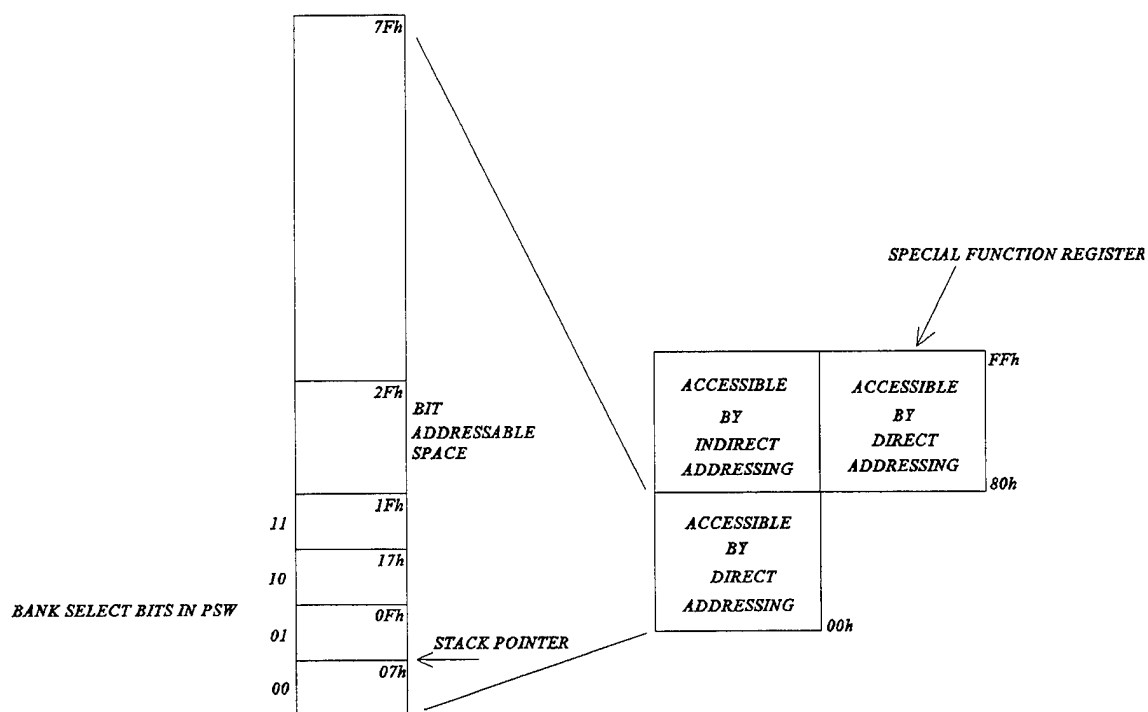
Figure 4.1   Internal Registers of the 80C51

Figure 4.2   Special Function Registers

2. B Register

The B register is used during multiply and divide operations. For other instructions, it can be treated as a scratchpad register. It resides at address F0h.

3. Program Status Word (PSW)

The eight bits of the Program Status Word register contain program status information as listed below. The PSW resides in address D0h.

Bit 0	Parity bit	Set/cleared by hardware each instruction cycle to indicate an odd/even number of "ones" in the Accumulator.
Bit 1	User Defined	
Bit 2	Over-Flow Flag	
Bits 3&4	RS0,RS1	Register bank select bits. Set/cleared by software to determine working register bank.

RS0/RS1	Selected Bank
0 0	Bank 0 (00h-07h)
0 1	Bank 1 (08h-0Fh)
1 0	Bank 2 (10h-17h)
1 1	Bank 3 (18h-17h)

Bit 5	Flag 0	Available to the user for general purposes.
Bit 6	Aux Carry Flag	Used for BCD operations
Bit 7	Carry Flag	

4. Stack Pointer (SP)

The 8-bit Stack Pointer is incremented before data is stored during PUSH and CALL executions. While the stack may reside anywhere in on-chip RAM, the Stack Pointer is initialized to 07h after a reset. This causes the stack to begin at location 08h.

5. Data Pointer (DPL, DPH)

The Data Pointer (DPTR) consists of a high byte (DPH) and a low byte (DPL). Its intended function is to store a 16-bit

28

address. It may be manipulated as a 16-bit register or as two independent 8-bit registers.

6. Ports 0 to 3

P0, P1, P2, and P3 are the SFR latches for Ports 0,1,2, and 3, respectively. Writing a one to a bit of any of these registers causes the corresponding port output pin to switch high. Writing a zero causes the output pin to switch low. When used as an input, the external state of a port pin will be held in the corresponding port register bit. More detail on the 80C51 ports can be found in Section 4.C.

7. Serial Data Buffer (SBUF)

The Serial Buffer is actually two separate registers, a transmit buffer and a receive buffer. When data is moved to SBUF, it goes to the transmit buffer and is held for serial transmission. Moving a byte to SBUF is what initiates the transmission. When data is moved from SBUF, it comes from the receive buffer. A more detailed description of the serial interface is located in Section 4.F.

8. Timer Registers Basic to 80C51

Register pairs TH0/TL0 and TH1/TL1 are the 16-bit counting registers for timer/counters 0 and 1 respectively. A more detailed description of the timer/counters is found in Section 4.E.

9. Control Registers

Special Function Registers IP, IE, TMOD, TCON, SCON, and PCON contain control and status bits for the interrupt system, the Timer/Counters, and the serial port. They are described in more detail in a later section.

C. PORT STRUCTURE AND OPERATION

All four ports of the 80C51 are bidirectional. Each consists of a latch (special function registers P0 through P3), an output driver and an input buffer. The output drivers of Port 0 and Port 2, along with the input buffer of Port 0, are used in 16-bit accesses to external memory. In this application, Port 0 outputs the low byte of the external memory address, time-multiplexed with the data byte being written or read. *ALE* (Address Latch Enable) should be used to enable an external latch to capture the low byte. Port 2 outputs the high byte of the external memory address when the address is 16 bits wide. During this time, the contents of the Port 2 SFRs are not modified and will reappear on the pins on the cycle following the memory access.

All four ports are multifunctional. Table 4.1 shows the alternate functions of each of the ports. The alternate functions can only be activated if the corresponding bit latch in the port SFR contains a one.

Figure 4.3 shows a functional diagram of a typical bit latch and I/O buffer in each of the four ports. The bit latch (one bit in the port's SFR) is represented as a D-type flip-flop, which will clock in a value from the internal bus in response to a "write to latch" signal from the CPU. The level of the port itself is placed on the internal bus in response to a "read pin" signal from the CPU. Some instructions that read a port activate the "read latch" signal and others activate the "read pin" signal.

The output drivers of Port 0 and Port 2 are switchable between the port SFR and the ADDR/DATA bus by an internal CONTROL signal for use in external memory accesses. During external memory accesses, the Port 2 SFR remains unchanged,

Port Pin	Alt. Name	Alternate Function
P0-0	A0	Address Bit 0
P0-1	A1	Address Bit 1
P0-2	A2	Address Bit 2
P0-3	A3	Address Bit 3
P0-4	A4	Address Bit 4
P0-5	A5	Address Bit 5
P0-6	A6	Address Bit 6
P0-7	A7	Address Bit 7
P1-0	T2	External clock input to timer/counter 2
P1-1	T2EX	Timer/counter 2 capture/reload trigger and direction control
P1-2	ECI	External count input to PCA
P1-3	CEX0	Ext I/O for PCA capture/compare Module 0
P1-4	CEX1	Ext I/O for PCA capture/compare Module 1
P1-5	CEX2	Ext I/O for PCA capture/compare Module 2
P1-6	CEX3	Ext I/O for PCA capture/compare Module 3
P1-7	CEX4	Ext I/O for PCA capture/compare Module 4
P2-0	A8	Address Bit 8
P2-1	A9	Address Bit 9
P2-2	A10	Address Bit 10
P2-3	A11	Address Bit 11
P2-4	A12	Address Bit 12
P2-5	A13	Address Bit 13
P2-6	A14	Address Bit 14
P2-7	A15	Address Bit 15
P3-0	RXD	Serial port input

Table 4.1   Alternate Port Pin Functions

Port Pin	Alt. Name	Alternate Function
P3-2	$\overline{INT0}$	External Interrupt 0
P3-3	$\overline{INT1}$	External Interrupt 1
P3-4	T0	External clock input for Timer 0
P3-5	T1	External clock input for Timer 1
P3-6	$\overline{WR}$	External Data Memory write strobe
P3-7	$\overline{RD}$	External Data Memory read strobe

Table 4.1 (cont'd)   Alternate Port Pin Functions

a. Port 0 Bit

b. Port 1 Bit

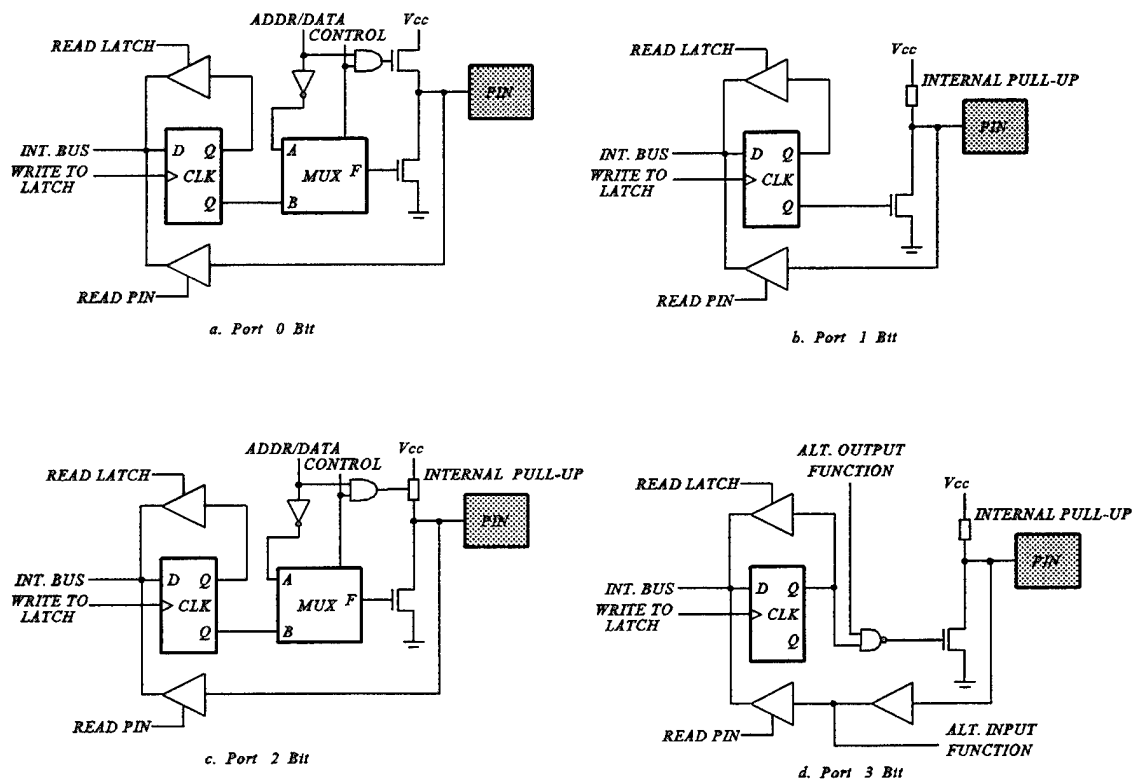c. Port 2 Bit

d. Port 3 Bit

Figure 4.3 80C51 Port Bit Latches

but the Port 1 SFR gets ones written to it. In Port 3, if the

bit latch contains a one, then the output level is controlled by the signal labeled "alternate output function". The actual pin level is always available to the pin's alternate input function, if any.

Ports 1, 2, and 3 have internal pull-ups and Port 0 has open drain outputs. Each I/O line can be independently used as an input or an output. However, Port 0 and Port 2 can not be used as general purpose I/O when being used as the ADDR/DATA BUS for external memory during normal operation. To be used as an input, the port latch must contain a one, which turns off the output driver FET. Then, for Ports 1, 2 and 3, the pin is pulled high by a weak internal pull-up and can be pulled low by an external source. Port 0 differs in that its internal pull-ups are not active during normal port operation. The pull-up FET is used only when the port is emitting a one during external memory accesses. Otherwise, the pull-up FET is off.

Consequently, Port 0 pins that are being used as output port lines are open-drain. Writing a one to the bit latch leaves both output FETs off, thus the pin floats. In this condition, it can be used as a high-impedance input. Because Ports 1, 2, and 3 have fixed internal pull-ups, they are sometimes called "quasi-bidirectional" ports. When configured as inputs, they pull high and will source current when externally pulled low. Port 0, on the other hand, is considered "true" bidirectional because, when configured as an input, it floats.

All the port latches have ones written to them by the reset function. If a zero is subsequently written to a port latch, it can be reconfigured as an input by writing a one to it.

During the execution of an instruction that changes the value in a port latch, the new value arrives during S6P2

(machine State 6, Phase 2) of the final cycle of the instruction. However, port latches are sampled by their output buffers only during Phase 1 of a clock period. During Phase 2, the output buffer holds the value it saw during the previous Phase 1. Consequently, the new value in the port latch won't actually appear at the output pin until the next Phase 1, which will be at S1P1 of the next machine cycle. The next section contains a more detailed description of the machine states.

The output buffers of Port 1, 2, and 3 can each drive 4 LS TTL inputs. Port 0 output buffers can each drive 8 LS TTL inputs. They do, however, require external pull-ups to drive NMOS inputs, except when being used as the ADDR/DATA BUS for external memory.

Some instructions that read a port read the latch and others read the pin. The instructions that read the latch rather than the pin are the "read-modify-write" instructions listed below.

```
ANL             Logical AND
ORL             Logical OR
XRL             Logical exclusive OR
JBC             Jump if bit = 1 and clear
CPL             Compliment bit
INC             Increment
DEC             Decrement
DJNZ            Decrement and jump if not zero
MOV PX.Y,C      Move carry bit to bit Y of Port X
CLR PX.Y        Clear bit Y of Port X
SET PX.Y        Set bit Y of Port X
```

The reason "read-modify-write" instructions are directed to the latch rather than to the pin is to avoid a possible misinterpretation of the logic level at the pin. For example, a port bit might be used to drive the base of a transistor. When a one is written to the bit, the transistor is turned on. If the CPU then reads the same port at the pin rather than the latch, it will read the base voltage of the transistor and

34

interpret it as a zero. Reading the latch rather than the pin
will return the correct value of one.

D. CPU TIMING

All 80C51 microcontrollers have an on-chip oscillator
which can be used, if desired, as the clock source for the
CPU. To use the on-chip oscillator, connect a crystal or
ceramic resonator between the *XTAL 1* and *XTAL 2* pins of the
microcontroller and capacitors from each pin to ground. The
value of the crystal is not particularly important as long as
it is less than the maximum clock speed of the 80C51 and
greater than about 3 MHz. Alternatively, an external clock
can be used by connecting it to the *XTAL 1* pin and leaving
XTAL 2 open (CMOS devices only).

A machine cycle consists of a sequence of six states,
numbered S1 through S6. Each state time lasts for two
oscillator periods. Thus, a machine cycle takes 12 oscillator
periods. Each state is divided into a Phase 1 and a Phase 2
half. Figure 4.4 shows the fetch/execute sequences in states
and phases for various kinds of instructions. Normally, two
program fetches are generated during each machine cycle, even
if the instruction being executed doesn't require it. If the
instruction being executed doesn't need more code bytes, the
CPU simply ignores the extra fetch and the Program Counter is
not incremented.

Execution of a one-cycle instruction (Figures 4.4a and
4.4b) begins during State 1 of the machine cycle, when the
opcode is latched into the instruction register. A second
fetch occurs during State 4 of the same machine cycle.
Execution is complete at the end of State 6 of this machine
cycle. The MOVX instructions take two machine cycles to
execute. No program fetch is generated during the second

cycle of a MOVX instruction. This is the only time program fetches are skipped. The fetch/execute sequence for MOVX instructions is shown in Figure 4.4d.

The fetch/execute sequences are the same whether the Program Memory is internal or external to the chip. Executiontimes do not depend on whether the Program Memory is internal or external.

Figure 4.5 shows the signal and timing involved in program fetches when the Program Memory is external. For external memory, the Program Memory read strobe ($\overline{PSEN}$) is normally activated twice per machine cycle. If an access to external Data Memory occurs, two $\overline{PSEN}$s are skipped, because the address and data bus are being used for the Data Memory
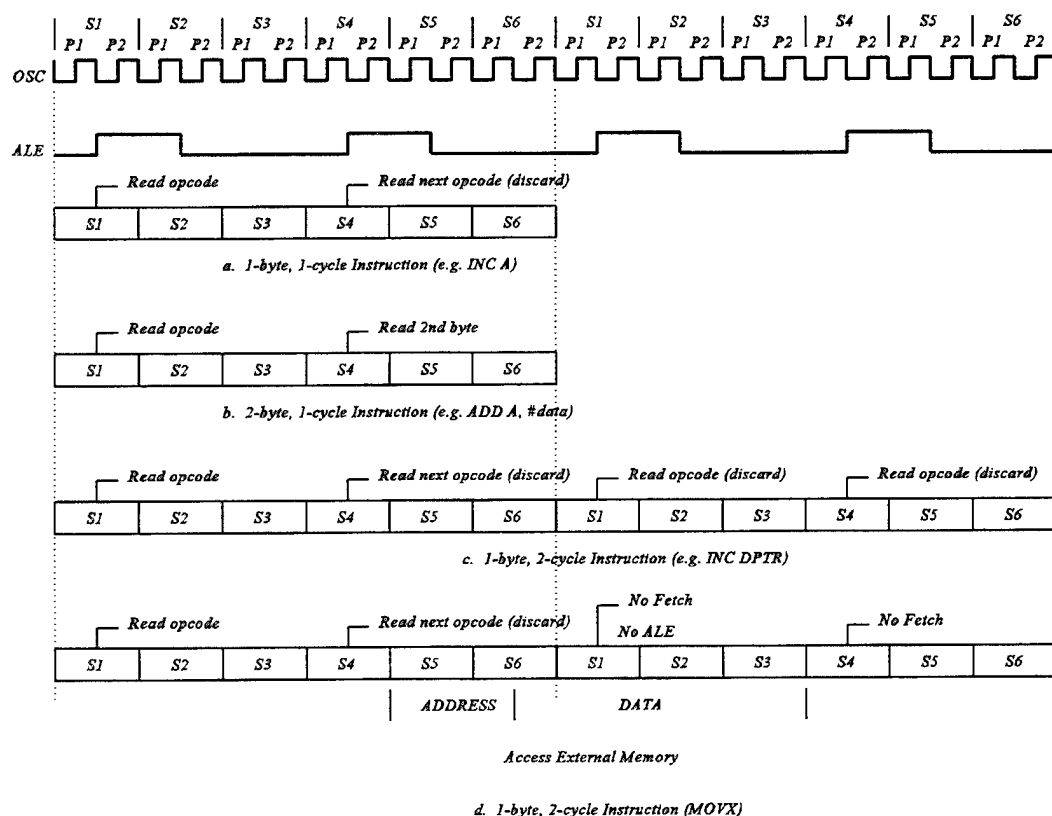
Figure 4.4   State Sequence For 80C51 Family Devices
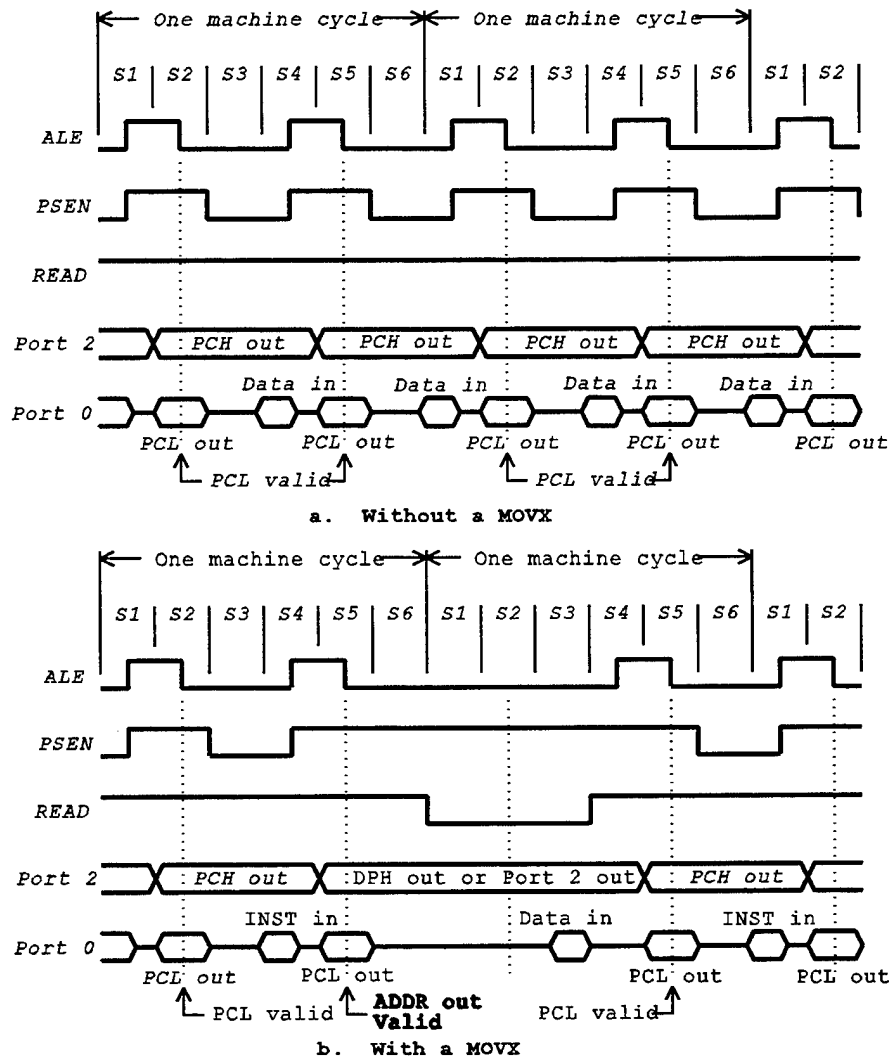
Figure 4.5 Bus Cycles in 80C51 Family Executing From External Memory

access. Note that a Data Memory bus cycle takes twice as much time as a Program Memory bus cycle. Figure 4.5 shows the relative timing of the addresses being emitted at Ports 0 and 2, and of $\overline{PSEN}$ and *ALE*. *ALE* is used to latch the low address byte from Port 0 into the external address latch.

When the CPU is executing code from internal Program Memory, $\overline{PSEN}$ is not activated and program addresses are not

37

emitted. However, *ALE* continues to be activated twice per machine cycle and is available as a clock output signal. Note, however, that one *ALE* is skipped during the execution of the MOVX instruction.

E. TIMER/COUNTERS

The 80C51 has two 16-bit Timer/Counter registers: Timer 0 and Timer 1. Both can be configured to operate either as timers or event counters. The TMOD and TCON special function registers configure and control the timers. TMOD selects one of four possible operating modes for both timer/counters and also provides some control functions. TCON is used to control the timer/counters. The high nibble of TMOD corresponds to Timer 1; the low nibble corresponds to timer 0. The four bits of TMOD per timer are used as follows:

MSB	GATE	When set, requires both INTx and TRx in TCON to enable the timer/counter. When cleared , only TRx is used as the enable.
	C/T	When set, counter function is selected. When cleared, timer function is selected.
LSB	M1/M0	Selects operating mode as follows:

    0 0   13-bit Timer. Lower five bits of "TLx" serves as 5-bit prescalar along with 8-bit "THx". This mode simulates an 8048 timer.

    0 1   16-bit Timer/counter. "THx" and "TLx" are cascaded; there is no prescalar.

    1 0   8-bit auto-reload Timer/counter. "THx" holds a value which is to be reloaded into "TLx" each time it overflows

    1 1   (Timer 0) TL0 is an 8-bit Timer/counter controlled by the standard Timer 0 control bits. TH0 is an 8-bit timer only controlled by Timer 1 control bits.
(Timer 1) Timer/counter 1 stopped. In this manner, the 80C51 can look like it has three timer/counters.

38

The bits of TCON are arranged in pairs instead of by nibble. Their function is defined as follows:

Bit 7   TF1   Timer 1 overflow flag. Set by hardware on overflow. Cleared by hardware when processor vectors to interrupt routine, or by clearing the bit in software.

Bit 6   TR1   Timer 1 Run control bit. Set/cleared by software to turn Timer/counter on/off

Bit 5   TF0   Timer 0 overflow flag.

Bit 4   TR0   Timer 0 Run control bit.

Bit 3   IE1   Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when the interrupt is processed.

Bit 2   IT1   Interrupt 1 Type control bit. Set/cleared by software to specify falling-edge/low-level triggered external interrupts.

Bit 1   IE0   Interrupt 0 Edge flag.

Bit 0   IT0   Interrupt 0 type control bit.

In the timer function, the register is incremented every machine cycle. A machine cycle consists of 12 oscillator periods, therefore the count rate is 1/12 of the oscillator frequency. In the counter function, the register is incremented in response to a 0-to-1 transition at its corresponding external input pin, T0 or T1. In this function, the external input is sampled during S5P2 of every machine cycle. When the samples show a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the register during S3P1 of the cycle following the one in which the transition was detected. Since it takes two machine cycles (24 oscillator periods) to recognize a 1-to-0 transition, the maximum count rate is 1/24 of the oscillator frequency. There are no restrictions on the duty cycle of the external input signal. However, to ensure that a given level is sampled at least once before it changes, it should be held for at least one full cycle.

F. SERIAL INTERFACE

The serial interface is full duplex, meaning it can transmit and receive simultaneously. It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the register. However, if the first byte still hasn't been read by the time reception of the second byte is complete, one of the bytes will be lost. The serial port receive and transmit registers are both accessed via the Special Function Register SBUF. Writing to SBUF loads the transmit register; reading SBUF accesses a physically separate receive register.

The serial port can operate in any of four modes. In MODE 0, the serial data enters and exits through *RXD*. *TXD* outputs the shift clock. Eight bits are transmitted/received (LSB first). The baud rate is fixed at 1/12 the oscillator frequency. In MODE 1, ten bits are transmitted (through *TXD*) or received (through *RXD*): a start bit (0), eight data bits (LSB first), and a stop bit (1). Upon reception, the stop bit goes into RB8 in Special Function Register SCON. The baud rate is variable. In MODE 2, 11 bits are transmitted (through TXD) or received (through RXD): a start bit (0), eight data bits (LSB first), a programmable ninth data bit, and a stop bit (1). On transmit, the ninth data bit (TB8 in SCON) can be used, for example, to send a parity bit (P in the PSW). On receive, the ninth data bit goes into RB8 in Special Function Register SCON while the stop bit is ignored. The baud rate is programmable to 1/32 or 1/64 times the oscillator frequency. MODE 3 is identical to MODE 2 in all respects except that the baud rate is variable. In all four modes, transmission is initiated by any instruction that uses SBUF as a destination register. Reception is initiated in MODE 0 by the condition RI = 0 and REN = 1. Reception is initiated in the other modes

by the incoming start bit if REN = 1.

MODE 2 and MODE 3 have a special provision for multiprocessor communications. In these modes, nine data bits are received. The ninth goes into RB8. The port can be programmed such that when the stop bit is received, the serial port interrupt will be activated only if RB8 = 1. This feature is enabled by setting bit SM2 in SCON. A way to use this feature in multiprocessor systems is as follows:

When the master processor transmits a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the ninth bit is one in an address byte and zero in a data byte. With SM2 = 1, no slave will be interrupted by a data byte. An address byte, however, will interrupt all slaves so that each slave can examine the received byte and see if it is being addressed. The addressed slave will clear its SM2 bit and prepare to receive the data bytes that will be coming. The slaves that were not being addressed leave their SM2s set and go about their business ignoring the coming data bytes.

SM2 has no effect in MODE 0 and, in MODE 1, can be used to check the validity of the stop bit. In a MODE 1 reception, if SM2 = 1, the receive interrupt will not be activated unless a valid stop bit is received.

The serial port control and status register is the Special Function Register SCON. This register contains not only the mode selection bits but also the ninth data bit for transmit and receive (TB8 and RB8) and the serial port interrupt bits (TI and RI). The contents of this register is as follows:

Bits 7/6   SM0/SM1   These bits determine the mode as
                     follows:

```
                              0 0   Mode 0    shift register
                              0 1   Mode 1    8-bit UART
                              1 0   Mode 2    9-bit UART
                              1 1   Mode 3    9-bit UART
        Bit 5      SM2        Enables      the     multiprocessor
                              communication feature in modes 2 and
                              3.   In mode 2 or 3, if SM2 is set
                              to  a  one,  then  RI  will  not  be
                              activated if the received ninth data
                              bit  (RB8)  is zero.   In Mode 1, if
                              SM2  =  1,  then  RI  will  not  be
                              activated if a valid stop bit was
                              not received.  In Mode 0, SM2 should
                              be zero.
        Bit    4 REN          Enables     serial     reception.
                              Set/cleared    by    software    to
                              enable/disable reception.
        Bit 3      TB8        The  ninth  data  bit  that will  be
                              transmitted in Modes 2 and 3.
        Bit 2      RB8        In Modes 2 and 3, contains the ninth
                              data bit received.   In Mode 1, RB8
                              contains the received stop bit.  RB8
                              is not used in Mode 0.
        Bit 1      TI         Transmit  interrupt  flag.   Set  by
                              hardware at the end of the eighth
                              bit  time  in  Mode  0,  or  at  the
                              beginning of the stop bit in the
                              other  modes  in  any transmission.
                              Must be cleared by software.
        Bit 0      RI         Receive  interrupt  flag.   Set  by
                              hardware at the end of the eighth
                              bit  time  in  Mode  0,  or  halfway
                              through the stop bit time in the
                              other   modes,   in   any  serial
                              reception.   Must  be  cleared  by
                              hardware.
```

G. INTERRUPTS

The 80C51 provides five interrupt sources. These are the two external interrupts, $\overline{INT0}$ and $\overline{INT1}$, two timer interrupts from TF0 and TF1, and the serial port interrupt. The external interrupts can be either level or transition sensitive depending on bits IT0 and IT1 in TCON. When an external interrupt is generated, the flag that generated it is cleared

by the hardware when the service routine is executed only if the interrupt was transition activated. If the external interrupt was level activated, then the external requesting source controls the request flag.

The timer interrupts, TF0 and TF1, are set by a rollover in their respective Timer/counter registers (except timer 0 in Mode 3). When a timer interrupt is generated, the flag that generated it is cleared by the on-chip hardware when the service routine is executed.

The serial port interrupt is generated by logically ORing RI with TI. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine will normally have to determine whether it was RI or TI that generated the interrupt and the bit will have to be cleared in software.

All of the bits that generate interrupts can be set or cleared by software, with the same result as though it had been set or cleared by hardware. That is, interrupts can be generated or pending interrupts can be canceled in software. Each of these interrupt sources can be individually enabled or disabled by setting or clearing a bit in Special Function Register IE, or all interrupts can be disabled at once. A one in any bit position enables that interrupt. The contents of IE are defined as follows:

```
Bit 7      EA     Disables all interrupts when cleared.
Bit 6             Not defined
Bit 5             Not defined
Bit 4      ES     Serial Port Interrupt Enable
Bit 3      ET1    Timer 1 overflow interrupt enable
Bit 2      EX1    External Interrupt 1 enable
Bit 1      ET0    Timer 0 overflow interrupt enable
Bit 0      EX0    External Interrupt 0 enable
```

Each interrupt source can also be individually programmed to one of two priority levels by setting or clearing a bit in Special Function register IP. A low priority interrupt can be

43

interrupted by a high priority interrupt but not by another low priority interrupt. A high priority interrupt can't be interrupted by any other interrupt source. If two interrupts of the same priority level are received simultaneously, an internal polling sequence determines which request is serviced. Thus, within each priority, there is a second priority structure determined by the polling sequence. External Interrupt 0 has the highest priority within this structure, followed by Timer/counter 0, external interrupt 1, timer/counter 1, and finally, the serial port at the low end of the priority spectrum. The individual bits within the IP register are defined as follows:

```
Bits 7-5        Not used
Bit 4     PS    serial port interrupt priority level
Bit 3     PT1   Timer 1 interrupt priority level
Bit 2     PX1   External interrupt 1 priority level
Bit 1     PT0   Timer 0 interrupt priority level
Bit 0     PX0   External interrupt 0 priority level
```

The interrupt flags are sampled at S5P2 of every machine cycle. The samples are polled during the following machine cycle. If one of the flags was in a set condition at S5P2 of the preceding cycle, the polling cycle will find it and the interrupt system will generate a LCALL to the appropriate service routine, provided this hardware-generated LCALL is not blocked by any of the following conditions: an interrupt of equal or higher priority already in progress, the current polling cycle is not the final cycle in the execution of the instruction in progress, and the instruction in progress is RETI or any write to the IE or IP registers.

The hardware-generated LCALL pushes the contents of the Program Counter on to the stack (but it does not save the Program Status Word) and reloads the PC with the service routine address. The vector addresses are as listed below:

SOURCEVECTOR ADDRESS

SOURCE	VECTOR ADDRESS
IE0	0003h
TF0	000Bh
IE1	0013h
TF1	001Bh
RI+TI	0023h

Execution proceeds from that location until the RETI instruction is encountered. The RETI instruction informs the processor that this interrupt routine is no longer in progress, then pops the top two bytes from the stack and reloads the program counter. Execution of the interrupted program continues from where it left off. Note that a simple RET instruction would have returned execution to the interrupted program but it would have left the interrupt control system thinking the interrupt was still in progress, making future interrupts impossible.

The external sources can be programmed to be level-activated or transition-activated by setting or clearing bit IT1 or IT0 in the TCON register. A one in ITx corresponds to edge sensitivity. In this mode, if successive samples of the INTx pin show a high in one cycle and a low in the following cycle, interrupt request flag IEx in TCON is set. IEx will be automatically cleared by the CPU when the service routine is called. If the external interrupt is level sensitive, the external source has to hold the request active until the requested interrupt is actually generated. Then it has to deactivate the request before the interrupt service routine is completed, or else another interrupt will be generated.

H. CONCLUSION

This chapter described in detail the architecture of the 80C51 family of microcontrollers. A thorough knowledge of the material in this chapter is imperitive in order to program the testbench efficiently. A thorough knowledge of the 80C51

family instruction set is also necessary in order to use the testbench. This material is provided in the next chapter.

.

V. 80C51 FAMILY INSTRUCTION SET

The information presented in this chapter was paraphrased from the 80C51 data sheets published by Philips Semiconductors in March of 1995.

A. OVERVIEW

The 80C51 instruction set is optimized for 8-bit control applications. It provides a variety of fast addressing modes for accessing the internal RAM to facilitate byte operations on small data structures. The instruction set provides extensive support for one-bit variables as a separate data type, allowing direct bit manipulation in control and logic systems that require Boolean processing.

B. ADDRESSING MODES

1. Direct Addressing

In direct addressing, the operand is specified by an eight-bit address field in the instruction. Only internal Data RAM and SFRs can be directly addressed.

2. Indirect Addressing

In indirect addressing, the instruction specifies a register which contains the address of the operand. Both internal and external RAM can be indirectly addressed. The address register for eight-bit addresses can be R0 or R1 of the selected bank, or the stack pointer. The address register for 16-bit addresses can only be the 16-bit "data pointer" register, DPTR.

3. Register Instructions

The register banks, containing registers R0 through R7,

can be accessed by certain instructions which carry a three-bit register specification within the opcode of the instruction. Instructions that access the registers this way are code-efficient because this mode eliminates an address byte. When the instruction is executed, one of the eight registers in the selected bank is accessed. One of four banks is selected at execution time by the two bank select bit in the PSW.

4. Register-Specific Instructions

Some instructions are specific to a certain register. For example, some instructions always operate on the Accumulator, Data Pointer, etc. and no address byte is needed to point to it. The opcode itself does that. Instructions that refer to the Accumulator as A assemble as Accumulator specific opcodes.

5. Immediate Constants

The value of a constant can follow the opcode in Program Memory. For example, MOV A,#100 loads the Accumulator with the decimal number 100. The same number could be specified in hex digits as 64h.

6. Indexed Addressing

Only Program Memory can be accessed with indexed addressing and it can only be read. This addressing mode is intended for reading look-up tables in Program Memory. A 16-bit base register (either DPTR or the Program Counter) points to the base of the table and the Accumulator is set up with the table entry number. The address of the table entry in Program Memory is formed by adding the Accumulator data to the base pointer.

Another type of indexed addressing is used in the "case jump" instruction. In this case, the destination address of a jump instruction is computed as the sum of the base pointer and the Accumulator.

C. ARITHMETIC INSTRUCTIONS

The menu of arithmetic instructions is listed in Table 5.1. The table indicates the addressing modes that can be used with each instruction to access the <byte> operand. For example, the ADD A,<byte> instruction can be written as:

```
ADD   A,7Fh      (direct addressing)
ADD   A,@R0      (indirect addressing)
ADD   A,R7       (register addressing)
ADD   A,#127     (immediate constant)
```

The execution times listed in the table assume a 12 MHz clock frequency. All of the arithmetic instructions execute in 1 µs except the INC DPTR instruction, which takes 2 µs, and the Multiply and Divide instructions, which take 4 µs. Note that any byte in the internal Data Memory space can be incremented without going through the Accumulator.

One of the INC instructions operates on the 16-bit Data Pointer. The Data Pointer is used to generate 16-bit addresses for external memory. Therefore, being able to increment it in one 16-bit operation is a useful feature. The MUL AB instruction multiplies the Accumulator by the data in the B register and puts the 16-bit product into the concatenated B and accumulator registers. The DIV AB instruction divides the Accumulator by the data in the B register and leaves the quotient in the Accumulator and the eight-bit remainder in the B register.

Oddly enough, DIV AB finds less use in arithmetic "divide" routines than in radix conversions and programmable shift operations. In shift operations, dividing a number by

2n shifts it n bit to the right. Using DIV AB to perform the division completes the shift in 4 μs and leaves the B register holding the bits that were shifted out. The DA A instruction

MNEMONIC	OPERATION	ADDRESSING MODES				EX. TIME
		DIR	IND	REG	IMM	(μs)
ADD A,<byte>	A=A+<byte>	x	x	x	x	1
ADDC A,<byte>	A=A+<byte>+C	x	x	x	x	1
SUBB A,<byte>	A=A-<byte>-C	x	x	x	x	1
INC A	A=A+1	Accumulator Only				1
INC <byte>	byte=byte+1	x	x	x		1
INC DPTR	DPTR=DPTR+1	Data Pointer Only				2
DEC A	A=A-1	Accumulator Only				1
DEC <byte>	byte=byte-1	x	x	x		1
MUL AB	B:A=BxA	ACC and B Only				4
DIV AB	A=Int[A/B] B=Mod[A/B]	ACC and B Only				4
DA A	Decimal Adj.	Accumulator Only				1

Table 5.1 Arithmetic Instructions

is for BCD arithmetic operations. In BCD arithmetic, ADD and ADDC instructions should always be followed by a DA A operation to ensure the result is also in BCD. The DA A operation produces a meaningful result only as the second step in the addition of two BCD bytes.

D.   LOGICAL INSTRUCTIONS

Table 5.2 shows the 80C51 logical instructions. The instructions that perform Boolean operations (AND, OR, Exclusive OR, NOT) on bytes perform the operation on a bit-by-

bit basis. That is, if the Accumulator contains 00110101b and the byte contains 01010011b, then ANL A,<byte> will leave the Accumulator holding 00010001b.

The addressing modes that can be used to access the <byte> operand are as follows for the ANL instruction:

```
ANL   A,7Fh      (direct addressing)
ANL   A,@R1      (indirect addressing)
ANL   A,R6       (register addressing)
ANL   A,#53h     (immediate constant)
```

All of the logical instructions that are Accumulator-specific execute in 1 μs (assuming a 12 MHz clock). The others take 2 μs.

MNEMONIC	OPERATION	ADDRESSING MODES				EXE.
		DIR	IND	REG	IMM	TIME
ANL A,<byte>	A=A AND <byte>	X	X	X	X	1
ANL <byte>,A	<byte>=<byte> AND A	X				1
ANL <byte>,#data	byte=byte AND #data	X				2
ORL A,<byte>	A=A OR <byte>	X	X	X	X	1
ORL <byte>,A	<byte>=<byte> OR A	X				1
ORL <byte>,#data	byte=byte OR #data	X				2
XRL A,<byte>	A=A XOR <byte>	X	X	X	X	1
XRL <byte>,A	<byte>=<byte> XOR A	X				1
XRL <byte>,#data	byte=byte XOR #data	X				2
CLR A	A = 00h	Accumulator Only				1
CPL A	A = NOT A	Accumulator Only				1
RL A	Rot. A left 1 bit	Accumulator Only				1
RLC A	Rot. left through C	Accumulator Only				1
RR A	Rot. A right 1 bit	Accumulator Only				1
RRC A	Rot. A right thru C	Accumulator Only				1
SWAP A	Swap nibbles in A	Accumulator Only				1

Table 5.2 Logical Instructions

Boolean operations can be performed on any byte in the
internal Data Memory space without going through the
Accumulator. The XRL <byte>,#data instruction, for example,
offers a quick and easy way to invert port bits, as in XRL
P1,#0FFh. If the operation is in response to an interrupt,
not using the Accumulator saves the time and effort to push it
onto the stack in the service routine.

The Rotate instructions (RL A, RLC A, etc.) shift the
Accumulator one bit to the left or right. For a left
rotation, the MSB rolls into the LSB position. The SWAP A
instruction interchanges the high and low nibbles within the
Accumulator. This is a useful operation for BCD
manipulations. For example, if the Accumulator contains a
binary number which is known to be less than 100, it can be
quickly converted to BCD by the following code:

```
MOVE B,#10
DIV  AB
SWAP A
ADD  A,B
```

Dividing the number by ten leaves the tens digit in the low
nibble of the Accumulator and the ones digit in the B
register. The SWAP and ADD instructions move the tens digit
to the high nibble of the Accumulator and the ones digit to
the low nibble.

E. DATA TRANSFER INSTRUCTIONS

1. Internal RAM

Table 5.3 shows the instructions that are available for
moving data around within the internal memory spaces and the
addressing modes than can be used with each one. With a 12
MHz clock, all of these instructions execute in either 1 or 2
μs.

The MOV <dest>,<src> instruction allows data to be

transferred between any two internal RAM or SFR locations without going through the Accumulator. Remember, the upper 128 bytes of data RAM can be accessed only by indirect addressing and SFR space only by direct addressing.

MNEMONIC	OPERATION	ADDRESSING MODES				EXE.
		DIR	IND	REG	IMM	TIME
MOV A,<src>	A=<src>	X	X	X	X	1
MOV <dest>,A	<dest>=A	X	X	X		1
MOV <dest>,<src>	<dest>=<src>	X	X	X	X	2
MOV DPTR,#data16	DPTR=constant				X	2
PUSH <src>	INC SP:MOV @SP,src	X				2
POP <dest>	MOV dest,@SP:DEC SP	X				2
XCH A,<byte>	A & byte exchanged	X	X	X		1
XCHD A,@Ri	A & @Ri exchange low nibbles		X			1

Table 5.3 Internal RAM Data Transfer Instructions

Note that in the 80C51 devices, the stack resides in on-chip RAM and grows upwards. The PUSH instruction first increments the Stack Pointer (SP), then copies the byte into the stack. PUSH and POP use only direct addressing to identify the byte being saved or restored but the stack itself is accessed by indirect addressing using the SP register. This means the stack can go into the upper 128 bytes of RAM, if they are implemented, but not into the SFR space.

The upper 128 bytes of RAM are not implemented in the 80C51 nor in its ROMless or EPROM counterparts. With these devices, if the SP points to the upper 128 bytes, PUSHed bytes are lost and POPed bytes are indeterminate.

The data transfer instructions include a 16-bit MOV that can be used to initialize the Data Pointer (DPTR) for look-up

tables in program memory or for 16-bit external accesses.

The XCH A,<byte> instruction causes the Accumulator and the addressed byte to exchange data. The XCHD A,@Ri instruction is similar but only the low nibbles are exchanged. To see how XCH and XCHD can be used to facilitate data manipulations, consider first the problem of shifting an eight-digit BCD number two digits to the right. This can be done with MOVs or by making use of XCH instructions as follows:

```
MOV A,2Eh              CLR A
MOV 2Eh,2Dh            XCH A,2Bh
MOV 2Dh,2Ch            XCH A,2Ch
MOV 2Ch,2Bh            XCH A,2Dh
MOV 2Bh,#0            XCH A,2Eh
```

Both methods require five lines of code but the use of the XCH instruction is more efficient. Doing the routine with direct MOVs uses 14 code bytes and 9 µs of execution time (assuming a 12 MHz clock). The same operation with XCHs uses only 9 bytes and executes in 5 µs, almost twice as fast.

2. External RAM

Table 5.4 shows the data transfer instructions that access external data memory. Only indirect addressing can be used. The choice is whether to use a one-byte address, @Ri, where Ri can be either R0 or R1 of the selected register bank, or a two-byte address, @DPTR. The disadvantage of using 16-bit addresses if only a few Kbytes of external RAM are involved is that 16-bit addresses use all eight bits of Port 2 as address bus. On the other hand, eight-bit addresses allow one to address a few bytes of RAM without having to sacrifice all of Port 2. All of these instructions execute in 2 µs with a 12 MHz clock. Note that in all external data RAM accesses, the Accumulator is always either the destination or source of the data.

The read and write strobes to external RAM are activated only during the execution of a MOVX instruction. Normally these signals are inactive and, if they're not going to be used at all, their pins are available as extra I/O lines.

ADDRESS WIDTH	MNEMONIC	OPERATION	EXE. TIME (μs)
8 Bits	MOVX A,@Ri	Read ext RAM @Ri	2
8 Bits	MOVX @Ri,A	Write ext RAM @Ri	2
16 bits	MOVX A,@DPTR	Read ext RAM @DPTR	2
16 Bits	MOVX @DPTR,A	Write RAM @DPTR	2

Table 5.4   External RAM Data Transfer Instructions

Table 5.5 shows two instructions that are available for reading lookup tables in program memory. Since these instructions access only program memory, the lookup tables can only be read, not updated. If the table access is to external program memory, then the read strobe is $\overline{PSEN}$.

MNEMONIC	OPERATION	EXE TIME
MOVC A,@A+DPTR	Read prog memory at (A+DPTR)	2
MOVC A,@A+PC	Read prog memory at (A+PC)	2

Table 5.5   Lookup Table Read Instructions

The mnemonic is MOVC for "move constant". The first MOVC instruction in Table 5.5 can accommodate a table of up to 256 entries. The number of the desired entry is loaded into the Accumulator and the Data Pointer is set to point to the beginning of the table. The MOVC A,@A+DPTR copies the desired table entry into the Accumulator.

The other MOVC instruction works the same way, except the

Program Counter (PC) is used as the table base and the table is accessed through a subroutine. First, the number of the desired entry is loaded into the Accumulator, then the subroutine is called:

```
MOV A,ENTRY NUMBER
CALL TABLE
```

The subroutine "TABLE" would look like this:

```
TABLE:   MOVC A,@A+PC
         RET
```

The table itself immediately follows the RET (return) instruction in program memory. This type of table can have up to 255 entries, numbered 1 through 255. Number 0 cannot be used because at the time the MOVC instruction is executed, the PC contains the address of the RET instruction. An entry numbered 0 would be the RET opcode itself.

F.   BOOLEAN INSTRUCTIONS

80C51 devices contain a complete Boolean (single-bit) processor. The internal RAM contains 128 addressable bits and the SFR space can support up to 128 addressable bits as well. All of the port lines are bit-addressable and each one can be treated as a separate single-bit port. The instructions that access these bits are not just conditional branches but a complete menu of move, set, clear, compliment, OR, and AND instructions. These kinds of bit operations are not easily obtained in other architectures with any amount of byte-oriented software. The instruction set for the Boolean processor is shown in Table 5.6. All bit accesses are by direct addressing.

Note how easily an internal flag can be moved to a port pin:

```
MOV   C,FLAG
MOV   P1.0,C
```

56

In this example, FLAG is the name of any addressable bit in the lower 128 or in SFR space. An I/O line (the LSB of Port 1, in this case) is set or cleared depending on whether the flag bit is 1 or 0.

The Carry bit in the PSW is used as the single-bit Accumulator of the Boolean processor. Bit instructions that refer to the Carry bit as C assemble as *carry-specific* instructions (CLR C, etc.). The Carry bit also has a direct address because it resides in the PSW register which is bit-addressable.

Note that the Boolean instruction set includes ANL and ORL operations but not the XRL (exclusive OR) operation. An XRL operation is simple to implement in software. Suppose, for example, it is required to form the exclusive OR of two bits:

C = bit1 XRL bit2.

The software to do this could be as follows:

```
MOV  C,bit1
JNB  bit2,OVER
CPL  C
OVER: (continue)
```

First, bit1 is moved to the carry. If bit2 = 0, then C now contains the correct result. That is, bit1 XRL bit2 = bit1 if bit2 = 0. On the other hand, if bit2 = 1, C contains the compliment of the correct result. It need only be inverted (CPL C) to complete the operation. This code uses the JNB instruction, one of a series of bit-test instructions which execute a jump if the addressed bit is set (JC, JB, JBC) or if the addressed bit is not set (JNC, JNB). In the above case, bit 2 is being tested and, if bit2 = 0, the CPL instruction is jumped over.

JBC executes the jump if the addressed bit is set and also clears the bit. Thus, a flag can be tested and cleared

in one operation. All the PSW bits are bit-addressable and
available to the bit-test instructions.

MNEMONIC	OPERATION	EXE TIME
ANL C,bit	C = C AND bit	2
ANL C,/bit	C = C AND (NOT bit)	2
ORL C,bit	C = C OR bit	2
ORL C,/bit	C = C OR (NOT bit)	2
MOV C,bit	C = bit	1
MOV bit,C	bit = C	2
CLR C	C = 0	1
CLR bit	bit = 0	1
SETB C	C = 1	1
SETB bit	bit = 1	1
CPL C	C = NOT C	1
CPL bit	bit = NOT bit	1
JC rel	Jump if C = 1	2
JNC rel	Jump if C = 0	2
JB bit,rel	Jump if bit = 1	2
JNB bit,rel	Jump if bit = 0	2
JBC bit,rel	Jump if bit=1; CLR bit	2

Table 5.6 Boolean Instructions

The destination address for these jumps is specified to
the assembler by a label or by an actual address in program
memory. However, the destination address assembles to a
relative offset byte. This is a signed (two's compliment)
offset byte which is added to the PC in two's compliment
arithmetic if the jump is executed. The range of the jump is
-128 to +127 program memory bytes, relative to the first byte

following the instruction.

G. JUMP INSTRUCTIONS

Table 5.7 shows the list of unconditional jumps with their respective execution times for a 12 MHz clock. The table lists a single "JMP addr" instruction but in fact there are three (SJMP, LJMP, and AJMP), which differ only in the format of the destination address. JMP is a generic mnemonic which can be used if the programmer does not care which way the jump is encoded.

The SJMP instruction encodes the destination address as a relative offset. The instruction is two bytes long, consisting of the opcode and the relative offset byte. The jump distance is limited to a range of -128 to +127 bytes, relative to the instruction following the SJMP.

The LJMP instruction encodes the destination address as a 16-bit constant. The instruction is three bytes long, consisting of the opcode and two address bytes. The destination address can be anywhere in the 64K program memory space.

The AJMP instruction encodes the destination address as an 11-bit constant. The instruction is two bytes long, consisting of the opcode, which itself contains three of the eleven address bits, followed by another byte containing the low eight bits of the destination address. When the instruction is executed, these eleven bits are simply substituted for the low eleven bits of the Program Counter. The high five bits remain the same. Hence, the destination has to be within the same two Kbyte block as the instruction following the AJMP.

In all cases, the programmer specifies the destination address to the assembler in the same way, as a label or as a

16-bit constant. The assembler will put the destination address into the correct format for the given instruction. If the format required by the instruction will not support the distance to the specified destination address, a "Destination out of range" message is written to the List file.

MNEMONIC	OPERATION	EXE TIME
JMP addr	Jump to addr	2
JMP @A + DPTR	Jump to A + DPTR	2
CALL addr	Call subroutine at addr	2
RET	Return from subroutine	2
RETI	Return from interrupt	2
NOP	No operation	1

Table 5.7   Unconditional Jumps

The JMP @A+DPTR instruction supports case jumps. The destination address is computed at execution time as the sum of the 16-bit DPTR register and the Accumulator. Typically, the DPTR is set up with the address of a jump table. In a five-way branch, for example, an integer 0 through 4 is loaded into the Accumulator. The code to be executed might be as follows:

```
MOV   DPTR,#JUMP TABLE
MOV   A,INDEX_NUMBER
RL    A
JMP   @A+DPTR
```

The RL A instruction converts the index number (zero through four) to an even number in the range zero to eight because each entry in the jump table is two bytes long:

```
JUMP TABLE:
        AJMP CASE 0
        AJMP CASE 1
        AJMP CASE 2
        AJMP CASE 3
        AJMP CASE 4
```

60

There is only a single "CALL addr" instruction listed in Table 5.7 but there are actually two of them, LCALL and ACALL, which differ in the format in which the subroutine address is given to the CPU. Call is a generic mnemonic which can be used if the programmer does not care which way the address is encoded.

The LCALL instruction uses the 16-bit address format and the subroutine can be anywhere in the 64-Kbyte program memory space. The ACALL instruction uses the 11-bit format and the subroutine must be in the same 2-Kbyte block as the instruction following the ACALL. In either case, the programmer specifies the subroutine address to the assembler in the same way, as a label or as a 16-bit constant. The assembler will put the address into the correct format for the given instructions.

Subroutines should end with a RET instruction which returns execution to the instruction following the CALL. RETI is used to return from an interrupt service routine. The only difference between RET and RETI is that RETI tells the interrupt control system that the interrupt in progress is done. If there is no interrupt in progress at the time RETI is executed, then the RETI is functionally identical to RET.

Table 5.8 shows the list of conditional jumps available to the 80C51 programmer. All of these jumps specify the destination address by the relative offset method and are limited to a jump distance of -128 to +127 bytes from the instruction following the conditional jump instruction. The user specifies to the assembler the actual destination address the same way as the other jumps, as a label or as a 16-bit constant.

MNEMONIC	OPERATION	ADDRESSING MODES				EXE TIME
		DIR	IND	REG	IMM	(µs)
JZ rel	Jump if A = 0	Accumulator only				2
JNZ rel	Jump if A ≠ 0	Accumulator only				2
DJNZ byte,rel	Decrement & jump if not 0	X		X		2
CJNE A,byte,rel	Jump if A ≠ byte	X			X	2
CJNE byte,#data,rel	Jump if byte ≠ #data		X	X		2

Table 5.8   Conditional Jump Instructions

There is no Zero bit in the PSW. The JZ and JNZ instructions test the Accumulator data for that condition.

The DJNZ instruction (Decrement and Jump if Not Zero) is for loop control. To execute a loop N times, load a counter byte with N and terminate the loop with a DJNZ to the beginning of the loop, as shown below for N = 10.

```
          MOV  COUNTER,#10
     LOOP: (begin loop)
          .
          .
          .
          (end loop)
          DJNZ COUNTER,LOOP
          (continue)
```

The CJNE instruction (Compare and Jump if Not Equal) can also be used for loop control. Two bytes are specified in the operand field of the instruction. The jump is executed only if the two bytes are not equal. Another application of this instruction is in "greater than/less than" comparisons. The two bytes in the operand field are taken as unsigned integers. If the first is less than the second, then the Carry bit is set. If the first is greater than or equal to the second, the

carry bit is cleared.

H.   CONCLUSION

The material in this chapter, along with a good working knowledge of the microcontroller architecture and testbench design, is sufficient to program the testbench to perform useful functions. In order to download and run programs, a good knowledge of the operating system is required. This material is presented in the next chapter.

VI. THE OPERATING SYSTEM

A. OVERVIEW

The operating system chosen for the testbench was written by Mr. Paul Stoffregen of Oregon State University (OSU). PAULMON, as it is called, is available as shareware from the OSU Electrical and Computer Engineering Department homepage (http://www.ece.orst.edu/~sllu/ece471f.html). It is a simple monitor program which provides enough on-line help to negate the need for much documentation. All of the information in this section comes from the PAULMON documentation written by Mr. Stoffregen.

PAULMON is designed exclusively for the 8051 family of microprocessors and requires the following hardware configuration:

★ External EPROM only; pin EA on the microprocessor must be connected to ground,
★ An 8K x 8 byte EPROM located at 0000h,
★ External RAM located at 2000h,
★ A single address space; signals $\overline{PSEN}$ and $\overline{READ}$ must be logically ORed together,
★ Communication to the user via the built-in UART of the microcontroller. Typically, a PC is used with a terminal program, an 8051 assembler, and a text editor,
★ A MAX232 or MAX233 serial line driver and receiver to interface the 80C51 to the PC must be used. No handshaking is used. The baud rate is chosen automatically.

B. FEATURES

1. Automatic Baud Rate Detection

This code was originally written by Mr. Kei-Yong Khoo. It is run immediately after a system reset. It waits for a <RETURN> character and uses it to calculate the timer #1 reload value. It requires only one character and stores the

reload value in four memory locations in internal RAM (78h, 79h, 7Ah, and 7Bh). These four locations are unlikely to be changed during the execution of a program or while the debugger is running. When another reset occurs, without removing the power, the program looks at these four locations. If all four agree, then it uses that reload value and does not require another keystroke. Occasionally, with crystal values which produce exact reload values (such as 7.3728 MHz), the baud rate detection routine may not correctly calculate the reload value. Garbage will get printed all over the screen. If this occurs, switch off the power and start over. The advantage of crystals such as the 7.3728 MHz is that they allow transmission speeds of 9600 and 19200 baud. It is highly recommended that the highest possible baud rate be used with this debugger as it tends to print quite a bit of text to the screen.

2. On-Line Help

By typing '?' at the main menu, a help screen summarizing the available commands is printed. On-line help is also available regarding the single-step execution feature. This help is accessed by typing '?' just after using the 'R' command. While in the single-step mode, a summary of commands is also available, again by typing '?'.

3. The <ESC> Key

The <ESC> key is supported extensively. It will abort all commands from any prompt. It will stop the list and hex dump commands in the middle of their printing. It will also interrupt the printing of text to the screen! This is useful at slow baud rates because a full screen of text can take quite a while to print at 300 baud.

4. The Download Program Command (type 'D')

This allows you to send the object code from the assembler to the external RAM. The object file must be a standard Intel hex format file, such as the .obj file created by the Pseudo-Assembler, by Pseudo-Corp. The file must be sent as an ASCII transfer. A protocol such as XMODEM is used. Pressing the <ESC> key at any time will abort the transfer. Please note that most communications programs use the <ESC> key to abort the transfer. In this case, the first <ESC> will halt the terminal; pressing it again will abort the receive process at the microcontroller. Unlike some other monitors, PAULMON will recognize the <ESC> key anywhere in the middle of the incoming data, not just at the beginning of a line.

5. The Run Program Command (Type 'R')

The run command allows the execution of a user program. Two types of run are supported, normal and single-step. The single-step mode is explained later, as it is fairly complex. During a normal run, the equivalent of an LCALL to the user code is given. During the execution of the program, the debugger has no control of the system unless the program calls one of the subroutines offered by the debugger in the jump table at location 0030h. After specifying which run mode is needed, a prompt appears for the location of the program to be executed with the current memory pointer value as the default choice. As is the case at all prompts, the <ESC> key will abort the run command. It is interesting to note that the run command leaves timer #1 in auto-baud rate generation mode. If serial communication is desired at the same baud rate as that used for the debugger, timer #1 need not be given a new reload value. It is recommended that the character input and output routines from the debugger be used via the jump table.

6. The New Memory Location Command (Type 'N')

The debugger operates with a pointer to the data memory with which you are working. This pointer is used by the list and hex dump command. It is also the default run location. The pointer is incremented as memory is viewed or modified. Just type 'N' to change it.

7. The List Command (Type 'L')

This debugger gives you the ability to list the program code directly from memory. All the 8051 mnemonics are supported, as well as the names of the special function registers. Bit addressable locations are displayed using the standard syntax (e.g. PWS.2 or 20.5) but individual bit location names are not supported (e.g. SCON.0 will print in place of RI). The original labels used in the source code cannot be printed. Instead, the memory locations are displayed. Other special Intel assembly formats, such as $ and CALL, are not supported. However, the list command can provide a reassuring look at the program directly from the memory.

8. The Hex Dump Command (Type 'H')

By typing 'H', the next 256 bytes of RAM are dumped to the screen in hex and ASCII. The <ESC> key must be pressed to abort the printout.

9. The Edit Command (Type 'E')

This command makes it possible to change the values of memory locations in the external RAM. The old value at each location is displayed. If <ESC> is pressed, the value at the current location is not changed.

10. The Jump Table

Despite the use of the word *jump*, the user must LCALL to these locations. The individual locations contain jumps to the subroutines, which all terminate with RET. The table provides the user with a memory location to call that WILL NOT CHANGE if the debugger is reassembled. The routines available are:

0030h:	Cout	Sends the byte in Acc to the serial port
0032h:	Cin	Waits for a character from the serial port, returns it in Acc
0034h:	pHex	Prints the two digit hex value in Acc to the serial port
0036:	pHex16	Prints the four digit hex value in DPTR to the serial port
0038:	pString	Prints the string in code memory pointed to by DPTR to the serial port. The string must terminate with 00h or a high bit set.
003Ah:	gHex	Gets a two digit hex value from the serial, returned in Acc.
003C:	gHex16	Gets a four digit hex value from the serial port, returned in DPTR
003Eh:	Esc	Checks to see if the <ESC> key is waiting in SBUF. Clears the buffer if it is and returns with carry set. Otherwise, it leaves SBUF untouched and returns with C = 0.
0040:	Upper	Converts character in Acc to uppercase if it is lowercase
0042:	Init	Automatic baud rate detection

The memory location can be placed directly in the code or an EQU statement can be used to make the code more readable. For example:

```
Program:  .EQU    gHex16, 003Ah  ;this makes the code
                                  nice
          MOV     DPTR, #StrLoc  ;Load DPTR
          LCALL   gHex16         ;print the DPTR
          MOV     A, #13
          LCALL   0030h          ;print a <RET>
          LCALL   0038h          ;print the string
          RET     StrLoc: .DB    "This is my String.",0
```

69

Most of these routines leave the registers unchanged. However, it is a good idea to consult the source code just to be sure. In particular the pHex routine destroys the contents of the accumulator.

11. Single-Step Mode

Documentation on this feature is not available. This feature has not been debugged completely. It works but beware of errors.

C. CONCLUSION

The previous four chapters provided specific information about the testbench, the 87C51 microcontroller, the instruction set, and the operating system respectively. This information is sufficient to program the testbench to perform any test within the design limitations. The next chapter contains a description of how to actually use the testbench to test two specific SRAM chips.

VII. CIRCUIT TESTING USING THE TESTBENCH

The testbench is intended to be used to test digital electronic circuits for sensitivity to Single Event Upsets (SEUs). To reiterate, SEUs are random changes in the stored logic value due to the impact of a subatomic particle. These particles are prevalent in the space environment, especially at geosynchronous altitudes. Any integrated circuit can be affected by SEUs, though digital logic storage elements such as registers and memories have a higher probability of experiencing an SEU. This testbench is primarily intended to test logic storage devices such as memory chips.

The basic testing algorithm for a memory chip is to write some data to each address location, then continually cycle through the addresses while reading the stored data. This is to be done while the chip is being irradiated. If an SEU occurs, the value read from the chip will be different from what was written. The address and the data value will then be sent to the operator via the serial port. Once the data value has been corrected, the algorithm continues until the next SEU is detected.

DRAMs present a special challenge to the testbench. These circuits must be refreshed continually or the data will fade away. Reading any address refreshes that address. When these memories are included in the circuit, a refresh circuit must be used. Every few milliseconds, this circuit refreshes the addresses either consecutively or a whole row at a time. During this time, the memory device is unavailable to the rest of the circuit. The testbench itself does not have any capability to generate refresh cycles. It may be possible to create a software-generated refresh signal by making use of the built-in timers of the 87C51 and the corresponding

interrupt. A second possibility is to build a refresh circuit on an adaptor board mounted in-line. A programmable 50% duty cycle clock can be programmed to come out of port 1, pin 0 of the microcontroller. It may be possible to use this signal to generate a periodic refresh cycle. Even without a refresh circuit, it may still be possible to test these devices on the testbench because the testing algorithm regularly cycles through all the addresses. The problem is to access all the addresses within the refresh period. Whether or not that is possible depends on the particular DRAM being tested and the efficiency of the code.

A. SETTING UP THE TESTBENCH

The following steps are required prior to performing any tests with the testbench. The operating system is loaded into the EPROM so it doesn't need to be reloaded. However, all other programs must be downloaded into RAM from the terminal. A portable 386 computer running Windows 3.1 is used as the terminal. All the programs required to operate the testbench are located in a directory called *tbench*. Chapter 6 contains more information on using the operating system.

1. Connect the Testbench to the Terminal and the UUT

A cable with a 25-pin sub-D RS-232 type connector on one end and a 9-pin sub-D connector on the other is required to connect CONN 0 of the testbench to the COMM1 port of the PC. The Unit-Under-Test (UUT) will have specific cabling requirements which are discussed in other sections.

2. Configure the Terminal

The Windows 3.1 terminal program should load automatically upon boot-up because the icon appears in the

startup menu. Open the terminal configuration file
tbench.trm. This defines the baud rate, parity bit, data
bits, and communication port. The portable PC has a maximum
baud rate of 9600 baud.

3. Establish Communications

Turn on the testbench. The monitor program needs a
return character to be sent in order to calculate the baud
rate and establish communications. Press return and the
welcome message should appear.

4. Download the Test Program

Type "d" to send a file to the testbench. A prompt will
appear asking for a .hex file. Press ALT-T to pull down the
transfer menu and select *Send Text File*. Select the desired
file and press return. It can be viewed by typing "L" and
specifying the location.

5. Run the Test Program

The test program can be run by typing "R" and providing
the memory location of the code. RAM begins at address 2000h,
thus this is the most likely addresswhere the program will
reside.

B. THE MOTOROLA 256 X 16 SRAM MEMORY CHIP

1. Description

The Motorola 256 X 16 SRAM chip tested as part of this
project is a gallium arsenide unit out of Motorola's DSP
component library. While still a proprietary device, Motorola
agreed to grant NPS researchers limited access to this design
in order to test the LT GaAs fabrication process. This
circuit was never originally intended to be packaged

separately, though. The circuit has insufficient output drivers to drive TTL inputs and uses input and output logic levels of 0 and 0.9-2.0 volts. To complicate matters further, the chip was not available in a package, only as an unpackaged die.

The chips were packaged in a 68-pin quad flat-pack by a third party vendor. This package turned out to be unusual and no chip carriers were available for it. Therefore, the challenge this chip presented was to design a chip carrier that could be placed in the particle beam. In addition, adaptor boards had to be fabricated to convert the input and output logic levels and to provide output drive capability.

2. Pin Assignments

The pin assignments are provided in Table 7.1 and the location of the pins is shown in Figure 7.1. Notice that there is no marker to designate pin 1 once the pins are cut from the pin die. This pin must be marked prior to cutting it loose.
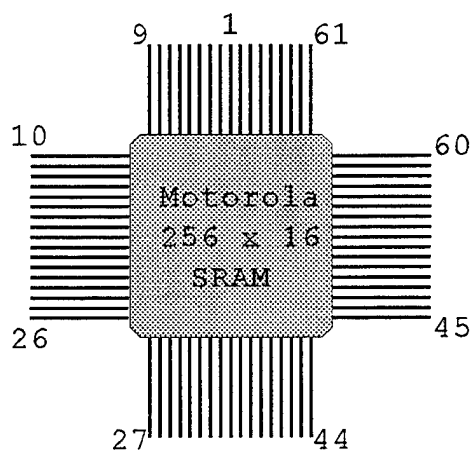

Figure 7.1 Motorola Pin Locations

PIN	USE	PIN	USE	PIN	USE	PIN	USE
1	N/C	18	N/C	35	N/C	52	N/C
2	Vdd-SRAM	19	WE	36	Vdd-DVR	53	D3-IN
3	Vdd-SRAM	20	Vdd-SRAM	37	D8-OUT	54	D4-IN
4	N/C	21	GND-SRAM	38	D9-OUT	55	D5-IN
5	N/C	22	A7	39	D10-OUT	56	D6-IN
6	Vdd-SRAM	23	A6	40	D11-OUT	57	D7-IN
7	GND-SRAM	24	GND-DVR	41	D12-OUT	58	D8-IN
8	A0	25	D0-OUT	42	D13-OUT	59	D9-IN
9	N/C	26	N/C	43	N/C	60	N/C
10	A1	27	D1-OUT	44	D14-OUT	61	D10-IN
11	A2	28	D2-OUT	45	D15-OUT	62	D11-IN
12	A3	29	D3-OUT	46	GND-DVR	63	D12-IN
13	A4	30	D4-OUT	47	Vdd-SRAM	64	D13-IN
14	A5	31	D5-OUT	48	GND-SRAM	65	D14-IN
15	GND-CLK	32	D6-OUT	49	D0-IN	66	D15-IN
16	CLK	33	D7-OUT	50	D1-IN	67	GND-SRAM
17	Vdd-CLK	34	Vdd-DVR	51	D2-IN	68	VDD-SRAM

Table 7.1   Motorola SRAM Pin Assignments

3.   Hardware Adapters

Logic level translators are required on both the input and output data buses and the address bus. The testbench uses TTL logic levels, i.e. 0-0.8V and 2.0-5V. The Motorola 4K SRAM uses 0V and 0.9-2V logic levels. In order to translate the logic levels, a string of two diodes and a resistor is used as shown in Figure 7.2. The diodes will drop approximately 0.7V each when turned on resulting in a 1.4V level for a logic one. This translation circuitry is placed
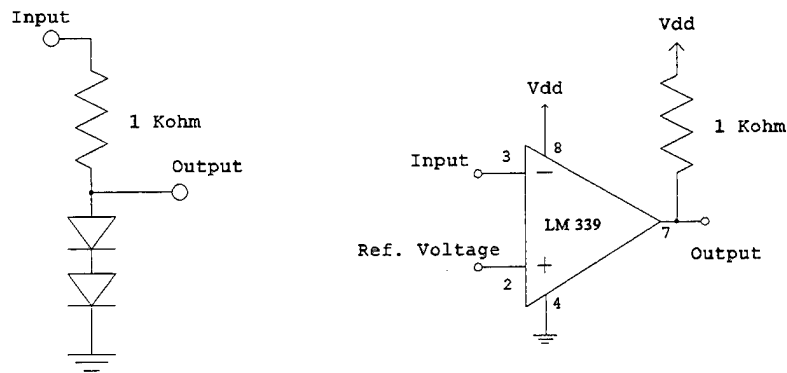
Logic Translation Circuitry   Output Drive and Logic translation Circuit

Figure 7.2 Logic Translation Circuitry Required For The Motorola 4K SRAM (after Stanley, 1989, pg 263)

on an adaptor board that mounts in-line, rather than as part of the testbench.

The output drivers that are provided with the chip were never intended to drive external circuitry since this device was never intended to be packaged separately. Therefore comparators are used on the output as shown in Figure 7.2 to provide both drive capability and logic translation.

The Motorola SRAM chip requires a data-in and a data-out bus as well as an address bus. Timing requirements of the SRAM eliminate the possibility of using internal address bus available at CONN 2. The chip requires a clock signal as well as the chip enable and write enable signals. The address and write enable lines must be changed when the clock is low. The address decoders and bit lines are precharged high during the clock-low period. The actual address is decoded when the clock is high. Data is either written (WE high) or read (WE low) during the high period of the clock. There is no clock signal available on the testbench so it must be simulated in software. Unfortunately, this is a multi-step process which negates the use of the address bus. The address bus of the SRAM is connected to CONN 3 with the adaptor board mounted in-line. A short 64-conductor ribbon connector is used between

the testbench and the adaptor board. A 50-conductor ribbon cable connects the adaptor board to the chip carrier board. The low byte of CONN 3 provides the address and the next two bits provide the write enable and clock signals. The DATA-IN Bus (to the SRAM) is connected to CONN 4 and the DATA-OUT Bus (from the SRAM) is connected to CONN 5. The adaptor boards are mounted in-line. IMPORTANT! It is physically possible to connect the chip carrier board directly to the testbench. Doing this will likely damage or destroy the Motorola SRAM chip due to the excess voltage.

4. Testing Algorithm

Figure 7.3 shows a flow chart for testing the Motorola SRAM chip. The actual code can be found in Appendix C.

C. THE VITESSE SRAM MEMORY CHIP

1. Description

The Vitesse SRAM chip is, essentially, the model VS12G422T IC but it has been packaged by a third-party vendor. It is important to note, however, that the pin assignments are not as originally specified by Vitesse. Vitesse originally used a standard 22-pin DIP package. The chips tested in this project were in a 28-pin DIP package. The actual pin assignments are provided in the next section.

The chip is a very high speed (5 to 8 ns access times), TTL-compatible SRAM fabricated using gallium arsenide. The chip is organized as 256 words, each of which has four bits. There are two chip selects, one active high and the other active low. It requires a single five-volt power supply.
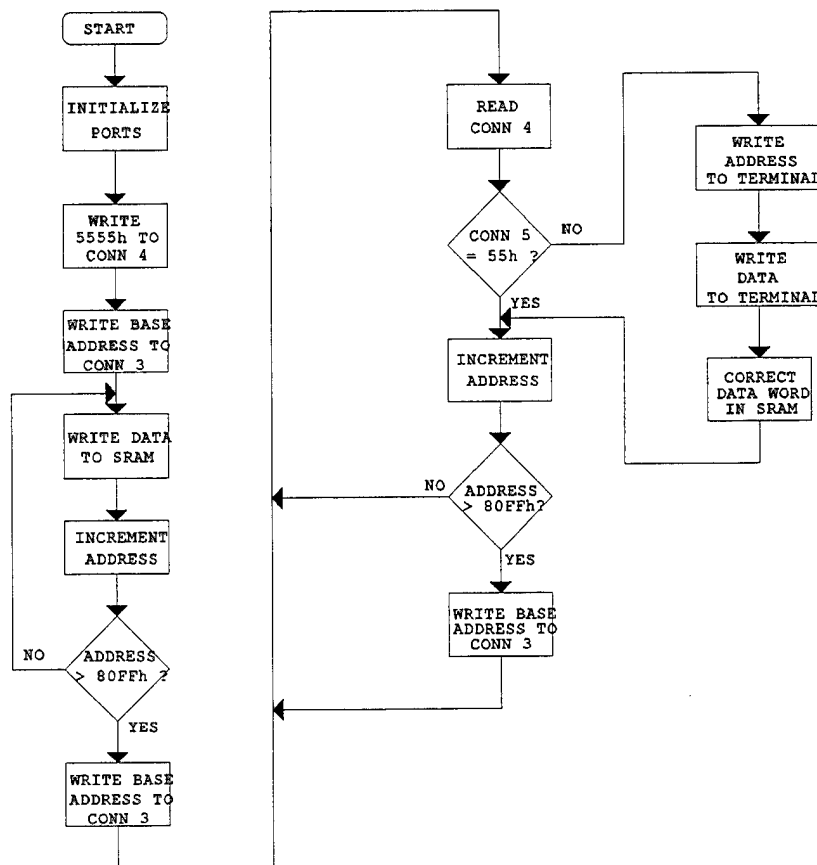
Figure 7.3 Flow Chart for Motorola SRAM

2. Pin Assignments

The pin assignments for the Vitesse SRAM are given in Table 7.2. These assignments are different than the original Vitesse VS12G422T SRAM package.

3. Testing Algorithm

The testing algorithm for the Vitesse SRAM is similar to but simpler than the Motorola algorithm. Each word in memory is only four-bits wide so the data buses can be interfaced directly to the microcontroller via CONN 1. Port 1 is used as the Data-in Bus (to the SRAM) and Port 0 is the Data-out Bus (from the SRAM). There are no difficult timing requirements to overcome, therefore the address bus is connected

78

PIN	USE	PIN	USE	PIN	USE	PIN	USE
1	DI-2	8	A-7	15	GND	22	CS2
2	GND	9	A-6	16	A3	23	Vcc
3	DO-1	10	A-5	17	A4	24	GND
4	DI-1	11	A-0	18	WE*	25	DO-3
5	Vcc	12	A-1	19	GND	26	DI-3
6	DO-0	13	A-2	20	CS1*	27	Vcc
7	DI-0	14	Vcc	21	OE*	28	DO-2

Table 7.2   Pin Assignments for Vitesse SRAM

directly to the microcontroller via CONN 2. Figure 7.4 shows
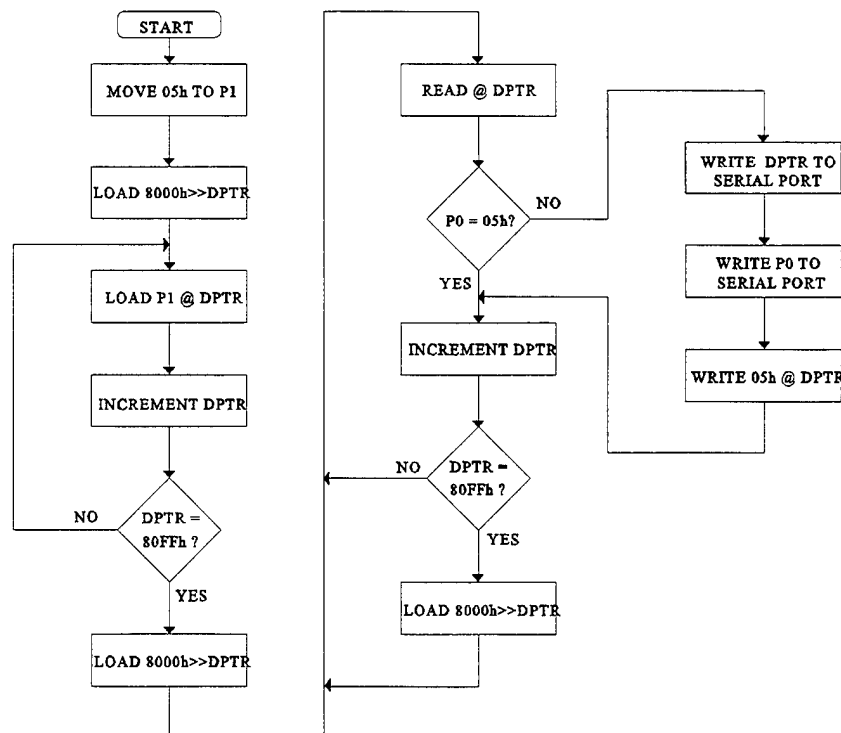the flow chart of the algorithm. The actual code is located

Figure 7.4   Flow Chart for Vitesse SRAM

79

in Appendix C.

VIII. CONCLUSION AND RECOMMENDATIONS

A. CONCLUSION

The testbench was completed and functionally tested just prior to the conclusion of this project. Assembly language programs were written to test the two memory chips involved in the LT GaAs project. While most of the original objectives were met, there is more work yet to be done. A self-test program would be nice to prove the functionality of the testbench. The testing algorithm for the two memory chips should be made more robust to better test the functionality of the chips rather than to just check the susceptibility to SEUs.

As is typical of research projects, numerous unforeseen problems arose and were individually solved during the construction of the testbench. Most of these problems involved the construction of the PC board. Wire-wrapping the board would have eliminated almost all of the problems and kept the project moving on schedule. However, the milled board adds a large measure of reliability to the testbench, looks much more professional, and provided a chance to learn how to build PC boards. The Naval Postgraduate School Physics Department owns the necessary equipment worth over $35000, but it is seldom used. In order to build this PC board, the entire process had to be figured out from scratch. A detailed description of the process is included as Appendix D in order to pass on the knowledge gained. The most important lessons on PC board layout are:

1. use the largest pad size possible to make soldering easier,

2. use traces that are as wide as possible to prevent them from breaking,

3. make all solder connections to the components on the back of the board,

4. if possible, double the mill width to further isolate the traces and prevent solder bridges, and

5. place all north-south runs on one side of the board and east-west runs on the opposite side.

B. RECOMMENDATIONS

The primary use of the testbench will be to test memory chips. A very important follow-on project would be to develop the capability to test DRAM chips. Whether a refresh circuit is necessary will depend on the refresh requirements of the particular DRAM chip and the efficiency of the code. It may not be necessary to use a refresh circuit if all the addresses are cycled through within the refresh period.

DRAMs are often constructed in a block format which requires both a row address and a column address. This square aspect ratio provides a possible method of avoiding the necessity of generating refresh cycles. An entire row can be refreshed whenever any address on that row is accessed. Thus, if the addresses are sequenced in order such that consecutive addresses are on different rows, it may be possible to test large DRAMs without the necessity of a refresh circuit.

If a refresh circuit is required, one possibility would be to use the i87C51 timer interrupts to periodically stop the test and refresh the memory in software. A second possibility would be to build an adaptor board to generate the refresh cycles. The output of a programmable timer in the i87C51 can be accessed at Port 1, pin 0. This can be used to initiate refresh cycles on an external adaptor board. Either way, adding the ability to test DRAMs would enhance the usefulness of the testbench significantly.

As mentioned earlier, programs need to be written to functionally test the memory chips. This test must be done before any meaningful test for SEU susceptibility can be done. This test would do more than just write fives to each address and then read each address. A functional test would have to write fives to each address first, then write As to change the bit pattern, then write 5s again to change back to the original bit pattern. This test checks to make sure that all bits can store both a zero and a one and that all bits can be set and cleared. The use of an alternating bit pattern will test for shorted leads. If this test passes, then a final test would be to write a different value to each address and verify that it was actually written. This checks for address independence. The functional test of the memory chip would be separate from the radiation test.

LIST OF REFERENCES

1. Amsler, D. E., *Design of a Universal Test Platform for Radiation Testing of Digital Components,* Master's Thesis, Naval Postgraduate School, Monterey, CA, Sep 1996.

2. Stanley, W. D., *Operational Amplifiers With Linear-Integrated Circuits*, 2nd Edition, Merrill Publishing Co., Columbus, OH, 1989.

3. ECE Dept. 8051 homepage, Paul Stoffregen, Oregon State University, http://www.ece.orst.edu/serv/8051/, accessed Dec 1996 to Feb 1997.

4. Weatherford, T. R., Marshall, P. W., Dale, C., Peczalski, A., Baier, S., McMorrow, D., Twigg, M., Campbell, A. B., "Soft Error Immune LT GaAs ICs," *18th Annual IEEE Gallium Arsenide Integrated Circuits Conference Proceedings*, Orlando, FL, Nov 3-6, 1996.

5. *Embedded Microcontrollers and Processors, Volume I*, Intel Corporation, 1992.

6. *Vitesse Product Data Book*, Vitesse Semiconductor Corporation, 1989.

7. *80C51 Family Harware Description*, Philips Semiconductors, 1995.

8. *80C51 Family Architecture*, Philips Semiconductors, 1995.

9. *TTL Logic Data Book*, Texas Instruments, 1988.

10. *MAXIM +5V-Powered, Multi-Channel RS-232 Drivers/Receivers*, Maxim Integrated Products, 1996.

11. *UT69RH051 Microcontroller Product Brief*, United Technologies Microelectronics Center, Inc., 1995.

APPENDIX A. LIST OF COMPONENTS

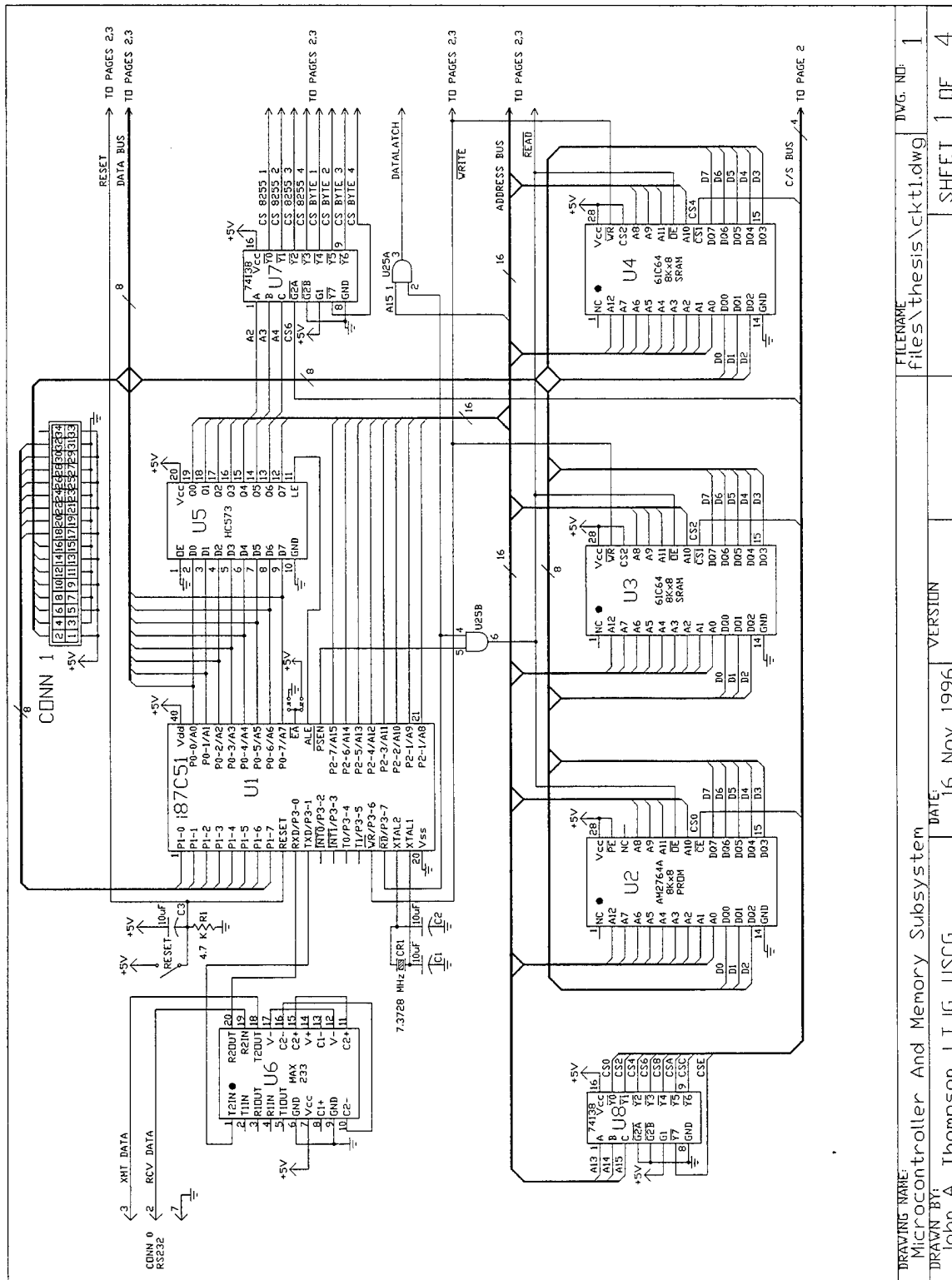|	RAD-HARD PART NUMBER	NON-RAD-HARD PART NUMBER	DESCRIPTION
1		MBK051010BW	Aluminum Enclosure By PFT Inc
1A1			Primary Circuit Board
1A1U1	UT69RH051	i87C51FC	8-bit 4-port microcontroller
1A1U2	UT28F64	AM2764A	8K x 8 EPROM
1A1U3	UT67164	61C64	8K x 8 RAM
1A1U4	UT67164	61C64	8K x 8 RAM
1A1U5	HCS573MS	74HC573	Broadside Octal Latch
1A1U6		MAX233	RS-232/TTL Converter
1A1U7	UT54ACS138	74HC138	3-to-8 Decoder
1A1U8	UT54ACS138	74HC138	3-to-8 Decoder
1A1U9	HS-82C55ARH	82C55A	Programmable Peripheral Interface
1A1U10	HS-82C55ARH	82C55A	Programmable Peripheral Interface
1A1U11	HS-82C55ARH	82C55A	Programmable Peripheral Interface
1A1U12	HS-82C55ARH	82C55A	Programmable Peripheral Interface
1A1U13	UT54ACS245	74HC245	Octal Bus Transciever
1A1U14	UT54ACS245	74HC245	Octal Bus Transciever
1A1U15	UT54ACS245	74HC245	Octal Bus Transciever
1A1U16	UT54ACS245	74HC245	Octal Bus Transciever
1A1U17	UT54ACS245	74HC245	Octal Bus Transciever
1A1U18	UT54ACS245	74HC245	Octal Bus Transciever
1A1U19	UT54ACS245	74HC245	Octal Bus Transciever
1A1U20	UT54ACS245	74HC245	Octal Bus Transciever
1A1U21	HCS574MS	74HC574	Broadside Octal D Flip-flop
1A1U22	HCS574MS	74HC574	Broadside Octal D Flip-flop

	RAD-HARD PART NUMBER	NON-RAD-HARD PART NUMBER	DESCRIPTION
1A1U23	HCS574MS	74HC574	Broadside Octal D Flip-flop
1A1U24	HCS574MS	74HC574	Broadside Octal D Flip-flop
1A1U25	UT54ACS08	74HC08	Quad Two-Input AND Gate
1A1C1			10uF Capacitor
1A1C2			10uF Capacitor
1A1C2			10uF Capacitor
1A1CR1			7.3728 MHz Crystal
1A1R1			8.4Kohm Resistor
1A1J1		AMP 102154-8	34-Pin Ejection-Style Pin Headers
1A1J2		AMP 1-102153-0	50-Pin Ejection-Style Pin Headers
1A1J3		AMP 1-102153-2	64-Pin Ejection-Style Pin Headers
1A1J4		AMP 1-102153-2	64-Pin Ejection-Style Pin Headers
1A1J5		AMP 1-102153-2	64-Pin Ejection-Style Pin Headers
1A2		LSWS-3031	3 Output Switching Power Supply by ACME Electric Corporation
1J0		AMP 745496-2	25-Pin Subminiature D Connector
1J1		3M 3329-0000	34-Pin Plug Connector
1J2		3M 3331-0000	50-Pin Plug Connector
1J3		3M ????-0000	64-Pin Plug Connector
1J4		3M ????-0000	64-Pin Plug Connector
1J5		3M ????-0000	64-Pin Plug Connector
1J6		108-0902-001	Banana Plug Receptacle-red BY EFJohnson
1J7		108-0902-001	Banana Plug Receptacle-red BY EFJohnson

RAD-HARD PART NUMBER	NON-RAD-HARD PART NUMBER	DESCRIPTION
1J8	108-0902-001	Banana Plug Receptacle-red BY EFJohnson
1J9	108-0903-001	Banana Plug Receptacle-Black BY EFJohnson
1SW1	1500R11E	Lighted Toggle Switch, SPST, Red, by EATON
1SW2	8551MZQE2	Push Button Switch, SPST, Momentary on, by C&K
1F1	3453-LF7	Fuseholder, Panel mount, by Littelfuse
1F2	3453-LF7	Fuseholder, Panel mount, by Littelfuse
1F3	3453-LF7	Fuseholder, Panel mount, by Littelfuse
1F4	3453-LF7	Fuseholder, Panel mount, by Littelfuse

APPENDIX B.   SCHEMATIC DIAGRAMS

This section contain the following schematic diagrams:

★   1A1 Microcontroller and Memory Subsystem

★   1A1 Address Ports, 16 and 32 Bit Ports

★   1A1 Peripheral Interface Subsystem, 8 to 32 Bit Ports

★   1A2 Power Supply

★   1A1 Board Layout - Component Location

TO PAGES 2,3
TO PAGES 2,3
TO PAGES 2,3
TO PAGES 2,3
TO PAGES 2,3
TO PAGE 2

RESET
DATA BUS
DATALATCH
WRITE
ADDRESS BUS
READ
C/S BUS

CONN 1

U1 i87C51

U5 HC573

U7 74138
CS 8255 1
CS 8255 2
CS 8255 3
CS 8255 4
CS BYTE 1
CS BYTE 2
CS BYTE 3
CS BYTE 4

U4 61C64 8K×8 SRAM

U3 61C64 8K×8 SRAM

U2 AM2764A 8K×8 PROM

U8 74138
CS0
CS2
CS4
CS6
CS8
CSA
CSC
CSE

U6 MAX 233

7.3728 MHz CR1

RESET R1 4.7 K C3 10uF

XMT DATA
RCV DATA

CONN 0 RS232

DRAWING NAME: Microcontroller And Memory Subsystem
DRAWN BY: John A. Thompson, LTJG, USCG
DATE: 16 Nov 1996
VERSION
FILENAME files\thesis\ckt1.dwg
DWG. NO: 1
SHEET 1 OF 4

Peripheral Interface Subsystem: 8 to 32 Bit Data Bus Adaptors

DRAWN BY: John A. Thompson, LTJG, USCG

FILENAME: c:\files\thesis\ckt3.dwg

DATE: 20 Nov 1996

SHEET 3 OF 4

DWG. NO: 1

97

DRAWING NAME:
1A2 Power Supply   ACME Model LSWT-3031

STUDENT NAME:
John A. Thompson, LTJG, USCG

DATE:
31 Jan 1996

FILENAME:
files\thesis\ckt4.dwg

DWG. NO: 1

SHEET 4 OF 4

CONN 5

82C55A
U9

74HC245
U13

CONN 3

74HC574   74HC574   74HC574   74HC574
U22   U21   U23   U24

74HC245
U14

RESET

C3   10 uF
R1
      8.4K

CONN 1

74HC138
U7

82C55A
U10

74HC245
U15

i87C51   74HC573
U1   U5

74HC08
U25

J1

74HC245
U16

MAX233
U6

AM2764A
U2

82C55A
U11

74HC245
U17

CONN 4

2
3
7   CONN 0

7.3228 Mhz

C1   10 uF
C2   10 uF

CONN 2

61C64
U3

74HC245
U18

82C55A
U12

74HC245
U19

61C64
U4

74HC138
U8

74HC245
U20

TEST BENCH PC BOARD   DRAWN BY: JOHN A. THOMPSON   4 DEC 1996

101

APPENDIX C.   SOURCE CODE

A.   MOTOROLA 256X16 SRAM

```
;*****************************************************************
;       Function RAMTEST
;       This function tests Motorola SRAMs by loading all
;       addresses with 0101010101010101 (5555h), and then
;       cycling through the addresses looking for any that have
;       changed value.  The address bus must be connected to
;       CONN 2.  The Data-in Bus (to the SRAM) must be
;       connected to CONN 4.  The Data-out Bus (from the SRAM)
;       must be connected to CONN 5.  The SRAM will be located
;       at 8000h and use CS8 for the enable.
;
;    .  Code written by:
;                  John A. Thompson, LT(jg), USCG
;                     Naval Postgraduate School
;                          2 Mar 1996
;*****************************************************************

;Equate Statements
Cout        equ  0030h      ;Sends Acc to serial port
pHex16      equ  0036h      ;Sends DPTR to serial port
Esc         equ  003Eh      ;Tests if escape has been pressed
U11PortA    equ  6008h      ;82C55 #3 Port A
U11PortB    equ  6009h      ;82C55 #3 Port B
U11PortC    equ  600Ah      ;82C55 #3 Port C
U11Control  equ  600Bh      ;82C55 #3 Control Word
Byte1       equ  6010h      ;Low byte of CONN 5
Byte2       equ  6014h      ;Low-Mid byte of CONN 5

org         2000h           ;Load function in RAM #1

;Load 5555h to all addresses
      ; Program the 82C55 Periphery Device #3 for output
            mov  DPTR,#U11Control ;82C55 #3 Control Word
                                  ;Address
            mov  A,#80h           ;All outputs, Mode 0
            movx @DPTR,A          ;Program 82C55 #3

      ; Write test word into output latches for CONN 4 bits
      ; 0-15
            mov  DPTR,#U11PortA ;82C55 #3 Port A Address
            mov  A,#55h            ;Load test word: 01010101b
            movx @DPTR,A
            mov  DPTR,#U11PortB ;82C55 #3 Port B Address
```

103

```
            movx @DPTR,A


        ; Set CONN 4 as output only and enable the output
            mov  DPTR,#U11PortC ;82C55 #3 Port C Address
            mov  A,#0003h       ;Enable outputs
            movx @DPTR,A


        ; Cycle through all addresses writing the test word
        ; NOTE:  The problem here is to generate the WRITE
        ; signal.  The testword is constantly output on CONN 4.
        ; In this situation, the Accumulator is written out,
        ; but goes nowhere.  But in doing so, a WRITE pulse
        ; is generated.
            mov  DPTR,#8000h    ;Load DPTR with base address
loop1:      movx @DPTR,A        ;Generate WRITE pulse
            inc  DPTR           ;Load next address
            cjne DPL,#00h,loop1 ;Test DPTR for last address


; Test SRAM for errors
; NOTE:  The Motorola SRAM chip uses a single W/R* signal.
; There is no necessity to generate a READ* signal; it is in
; the read mode by default.  However, it is necessary to
; generate a DATALATCH signal which is the result of ANDing
; A15 with READ*.
loop2:      mov  DPTR,#8000h    ;Load DPTR with base address
            mov  A,@DPTR        ;Generate a DATALATCH signal
loop3:      push DPTR           ;Save SRAM address
            mov  DPTR,#Byte1    ;Load address of low byte of
                                ;CONN 5
            mov  A,@DPTR        ;Read low byte
            cjne A,#55h,error   ;Test for error
            mov  DPTR,#Byte2    ;Load address of high byte of
                                ;CONN 5
            mov  A,@DPTR        ;Read High byte
            cjne A,#55h,error   ;Test for error
            pop  DPTR           ;Restore SRAM address
loop4:      inc  DPTR           ;Increment DPTR
            lcall ESC           ;Test for exit condition
            jbc  C,exit
            cjne DPL,#00h,loop3 ;Test DPTR for last address
            mov  A,#11001100    ;Load cycle-complete signal
            lcall Cout          ;Print it
            jmp  loop2


;Error Handling Routine
error:      mov  DPTR,#Byte1    ;Load address of low byte
            mov  A,@DPTR        ;Read low byte
            lcall Cout          ;Write erroneous data to
```

```
                            ;serial port
          mov   DPTR,#Byte2   ;Load address of high byte
          mov   A,@DPTR       ;Read High byte
          lcall Cout          ;Write erroneous data to
                             ;serial port
          pop   DPTR          ;Load Address
          lcall pHex16        ;Write address to serial port
          movx  @DPTR,A       ;Generate WRITE signal to
                             ;restore data, contents of
                             ;Accumulator go nowhere

          push  DPTR
          jmp   loop4         ;Return to SRAM test

;Exit Routine
exit:
          ret
          END
```

B.   VITESSE 256X4 SRAM

```
;*******************************************************
;     Function RAMTEST1
;     This function tests Vitesse SRAMs by loading all
;     addresses with 0101 (5h), and then cycling
;     through the addresses looking for any that have
;     changed value.  The address bus must be connected to
;     CONN 2 and the data buses connected to CONN 1.  Port 0
;     will be used for the data-in bus (to the SRAM) and
;     Port 1 will be used for the Data-out bus (from the
;     SRAM).  The SRAM will be located at 8000h and use CS8
;     for the enable.  The OE* pin on the SRAM is connected
;     to READ* and the WE* signal is connected to WRITE*.
;
;     Code written by:
;               John A. Thompson, LT(jg), USCG
;               Naval Postgraduate School
;                    10 Feb 1996
;*******************************************************

;Initialize System
Cout      equ   0030h    ;Sends Acc to serial port
pHex16    equ   0036h    ;Sends DPTR to serial port
Esc       equ   003Eh    ;Tests if escape has been pressed

org       2000h          ;Load function in RAM #1

;Load 0101 to all addresses
```

```
            mov    DPTR,#8000h     ;Load DPTR with base address
            mov    A,#05h          ;Load test value
loop1:      movx   @DPTR,A         ;Load data to SRAM
            inc    DPTR            ;Increment DPTR
            cjne   DPL,#00h,loop1  ;Test DPTR for last address

;Test SRAM for errors
;NOTE:  The data coming from the SRAM is on the low byte of
;Port 1.
loop2:      mov    DPTR,#8000h     ;Load DPTR with base address
loop3:      movx   A,@DPTR         ;Generate a READ cycle; data
                                   ;is useless
            movx   A,P1            ;Load the real data
            anl    A,#0Fh          ;Clear high byte
            cjne   A,#05h,error    ;Test for error
loop4:      inc    DPTR            ;Increment DPTR
            lcall  ESC             ;Test for exit condition
            jbc    C,exit
            cjne   DPL,#00h,loop3  ;Test DPTR for last address
            mov    A,#11001100     ;Load cycle-complete signal
            lcall  Cout            ;Print it
            jmp    loop2

;Error Handling Routine
error:      lcall  pHex16          ;Write address to serial port
            lcall  Cout            ;Write erroneous data to
                                   ;serial port
            mov    A,#05h          ;Restore correct data to SRAM
            movx   @DPTR,A
            jmp    loop4           ;Return to SRAM test

;Exit Routine
exit:
            ret
            END
```

APPENDIX D.   PCBOARD LAYOUT AND FABRICATION DETAILS

A.   OVERVIEW

The printed circuit board in the test bench was built locally at the Naval Postgraduate School. It is made from a .062" double-sided, nickel-plated blank pc board.   The outlines of the circuit traces and pads are machined into this board to isolate them from the remainder of the circuit. The remaining plating material forms the ground plane on one side and the power plane on the other. This eliminates the need to run power and ground lines throughout the circuit. It also provides inherent capacitance between the two planes which helps minimize noise. The basic steps followed to create the board are: 1) layout the circuit, 2) edit the file using a gerber editor, 3) create an outline file for each side of the board and a drill file, and 4) fabricate the physical board. This section does not include details of mounting the components or functionally testing the board

The circuit can be laid out using any CAD program that can generate an output file in gerber (.gbr) format. At the Naval Postgraduate School, *Cadence* and *Mentor Graphics* are available on the workstations. These powerful programs have a very steep learning curve but, once mastered, can provide error-free boards very quickly. There are also several PC-based programs available such as *Hiwire* and *Easytrax*. This circuit was laid out using Easytrax. This program is available, along with the reference manual and tutorial, as shareware from Protel Technology, Inc. It is a DOS utility that fits uncompressed on a single 3.5" floppy disk, making it very portable. It will run on any IBM PC, XT, AT, PS/2 and clones. The test bench was laid out using an IBM AT with a math coprocessor and the program worked well. Easytrax is

fully capable of building multi-layered boards and even includes some auto-routing capability. It does not, however, accept a netlist from which to lay out the circuit. Protel provides a library of component footprints with the Easytrax package. There are two programs which are used to create the board. *Easyedit* is used to actually lay out the traces and pads of the circuit. *Easyplot* is used to create the gerber (.gbl for gerber bottom layer and .gtl for gerber top layer) files for the top and bottom of the board. It is also used to print or plot the design. Easytrax is a stripped-down version of Protel's full-featured layout tool, *Autotrax*.

This program does not have the capability to combine segments of traces into one unified trace. This may lead to problems in later stages. If a trace has been started accidentally, it is important to use escape to cancel it rather than just terminating it on the starting point. Otherwise, zero-length traces will result and will cause problems later on. The wise designer will lay out the trace from beginning to end in one segment or, if necessary, place the trace in multiple segments ending each segment on a pad or a via.

It is very important to plan ahead before beginning the layout process. For example, the milling machine used for the testbench has a board-size capacity of 12" x 9". At least half an inch must be left around all the edges so the actual design size must be no larger than 8" x 11". The testbench PC board barely met this criteria. Also, the boards must be mounted in landscape mode so it is best to lay it out in this way rather than try to rotate it later on. Larger boards can be fabricated using the Space Systems Academic Group's milling machine.

The default pad and via size in Easytrax is .062" (62

mil). This pad size is very small for hand soldering. Using a larger pad size (75 - 85 mil) whenever possible is highly recommended. Most electronic components are built around a 100 mil (0.1") grid. Unless traces must run between pads, the pad size can be as large as 90 mil (assuming 10 mil bit size). The thermal pads are not really needed if there are other traces in the area to break up the surface plane. The default trace size of 12 mils should also be increased wherever possible. A 12 mil trace is so delicate that any contact with a soldering iron is likely to result in a broken trace. These breaks usually occur right at the pad. Merging the trace into the pad (using tear-drop shaped pads) will help prevent these breaks. If possible, it would save a lot of problems later on if double milling is done. The milling machine owned by the physics department can etch a 10 mil groove around the traces and pads. Doubling this width will make for much easier soldering in the future. This is done by making a second pass around the trace rather than by using a larger bit size.

The gerber editor used for the testbench is *PCGerber*, produced by CAD Solutions Inc. This full-featured program allows the designer to view and modify the design as may be needed (errors are not uncommon when the outline editor is run). CAD Solutions Inc. has a shareware gerber editor available via their homepage named *CAMWare* which is actually a full-featured editor with only its database size restricted. PCGerber creates a design file (.dsn) for the board which includes the aperture file, the board layer gerber files, and other related files such as the outline gerber files. Prior to opening the design file, an aperture file must be built which corresponds to the apertures used by the layout tool. It is often easier to just use an existing file and change the few Dcodes that are used in the design. In the design of the

testbench, thermal-relief pads were substituted for all the ground and power pads. These pads allow the designer to solder the pad without having to heat a large section of the board. To do this while laying out the circuit, all pads that were to be tied to ground were defined using a non-standard size pad. This was also done with the pads which were to be tied to V_{DD}, although a different pad size was used. Once the gerber files were loaded into PCGerber, the thermal-relief pads could be substituted simply by switching the code for the atypical-sized pads with the code for the thermal pad. There are many other refinements that can be made, such as radiusing all the trace corners or making the trace merge into the pad, resulting in a tear-drop shaped pad.

Prior to milling the board, the traces and pads must be converted to outlines. The milling machine actually cuts the outlines, not the traces. The outline gerber files for the testbench were created using *Protoboard 2.00* by Regulus Systems. This package was also used to drive the milling machine later on. To use this program, the input gerber files must be in the input directory (c:\pboard\input) and must be named L1.GBR, L2.GBR, etc. L1.GBR is the top of the board, and L2.GBR is the bottom of the board for a two-sided board. If the program runs to conclusion (this took over two hours using a 386 with 16 Mbytes of RAM and a math coprocessor), the output files will be in the output directory (c:\pboard\output). They will be named L1OUT.GBR and L2OUT.GBR for the top and bottom of the board, respectively. There will also be a drill file for each side (D1OUT.GBR, D2OUT.GBR, etc) and some files named L1TXT.GBR, L2TXT.GBR, etc. The two drill files should be identical unless some of the pads are not replicated on both sides. This would occur when a component is directly connected to the power or ground

plane instead of through thermal-relief pads. The L1TXT.GBR
file is the same as the input file L1.gbr but it has been
reordered and cleaned up a bit.

More than likely, an error will occur (unless you happen
to be using Cadence or some other program more powerful than
Easytrax) and the outline editor will hang up in an endless
loop.   In this case, the program must be halted and
modifications made before trying again.   There is a file
called LOUT.GBR located in PCGerber's working directory
(c:\pboard\ex1) which is the interim output gerber file. This
is the file that Protoboard writes to while it is creating the
outlines. If the program hangs up in an endless loop, this
file can get abnormally large. The finished bottom layer for
the   testbench was approximately 600 Kbytes in size.
Preliminary efforts resulted in files that exceeded 4 Mbytes
due to the endless loops. These can be difficult to edit in
the gerber editor because the editor recreates everything that
the outline generator did. To edit this file, copy it to the
directory containing PCGerber (c:\cam), and activate PCGerber
(by typing "cam").   Overlay the LOUT.GBR file on the
corresponding gerber file; the error is basically where the
outline stops. The cause of the error may not be apparent but
simply deleting the offending trace or pad and retyping it
should fix it. In the construction of the testbench PC board,
there were numerous zero-length traces which caused Protoboard
to fail. These showed up as dots, but the entire trace must
be deleted and the circuit redrawn to find them. In fact, so
many errors occurred that it was less difficult to simply
delete every trace and redraw them. The easiest way to do
this is to overlay both L1.gbr (for example) and L1TXT.GBR
which should both be identical. Make one a reference layer
and use it to show the proper location of all the traces. In
this fashion it is possible to delete all the traces in the

111

active layer and redraw them without having to engage any thought process at all. Of course, use the modified layer as the input for the next run of the outline converter.

The milling machine used for the testbench is part of the Protoboard package. The actual machine is a model PB800 which allows for a 9" by 12" board (8" x 11" realizable). This machine will perform all the milling, drilling, and outline cutting of the circuit. It also will perform auto-swaging to connect both sides of a via using manually inserted eyelets. This package is owned by the Physics Department at the Naval Postgraduate School and is located in the basement of Halligan Hall (656-2065). To use the milling machine, the outline gerber files and the drill file must be located in Protoboard's output directory (c:\pboard\output). Typing "p2" starts the program to mill the board. At the bottom of the screen is a prompt that provides guidance on what to do next.

It is probably best to run the drill file first since it is the smallest. Also, if milling is done first followed by the drilling, drilling through a thin piece of metal can twist the metal off the surface of the board. Furthermore, drilling before milling makes it easier to check layer-to-layer registration. Any offsets that may be required can be determined at this stage by drilling into a spare backing plate and measuring the position relative to the edges. Once the blank PC board is in place, the depth of the bit must be set so that the drill barely penetrates into the backing plate.

Once the holes are drilled and the routing bit installed, the top of the board can be milled. The depth of the bit can be set by making test cuts in an unused corner and measuring the width of the cut with an optical loupe. It might be wise to make test cuts in all four corners to ensure that the board is flat on the machine. It might actually be necessary to

slip paper under a corner to shim up low spots.

The file for the bottom of the board must be mirrored prior to running the program. An option is available that will do this automatically and effortlessly. Once the bottom of the board is milled, the eyelets can be installed in all the vias manually and then swaged (if desired) by the PB800 machine. This is done by inserting the eyelets from the bottom of the board and remounting the board with the top side up (the heads of the eyelets will be on the bottom). A special swaging bit is used with the depth set so that the bit just swages the eyelet without cutting it off. The drill file is run again to swage all the eyelets.

The testbench PC board took about five hours to complete all the milling, not including wasted time due to learning the intricacies of the equipment. Once the board is completed with all parts mounted and functionality tested, it would be wise to protect the board by spraying it with conformal coating. This is especially wise if a power and ground plane is used as in the test bench. The coating will help protect against shorts, especially if the board is being tested prior to mounting in an enclosure.

B. STEPS TAKEN DURING CONSTRUCTION OF DOUBLE-SIDED PCBOARD

1. Download Easytrax package from Protel Technology Inc.'s homepage (http://www.protel.com/download.htm) and install on a PC. The tutorials provided with the package are sufficient to teach the program within an hour or two.

2. Plan ahead prior to laying out the board. The milling machine used for the testbench has a capacity of 8"x11" and the board must be laid out in landscape mode. This is good information to have prior to laying out the board. Type *easyedit* from within the Easytrax directory to enter the editor. Lay out the circuit. Begin and end all traces on pads or vias. Multiple segment traces will cause problems later on. Place all traces heading north-south on one side of the board and those heading east-west on the other. Make as many connections to the components as possible on the opposite side of the board because the component will block access for soldering on the top of the board. On the testbench, the top of the board is the ground plane and the bottom is the power plane. Thermal-relief pads were used in order to make soldering easier. These pads have a regular pad on the side that must be isolated and a partial outline of the pad on the other. In order to insert these later on in the gerber editor, these pads were given an odd-ball size (.040" for ground and .050" for power). All the other pads and vias are .062". However, .062" proved to be too small for convenient soldering. Make all the pads as large as possible, up to about .090". Also, use as large a trace as possible within reason and miter the corners. Eliminating hard corners in the traces becomes more important as the frequency increases. It

might become necessary for high frequency circuits to make sure all the traces in a bus are exactly the same length by snaking the closest runs. This will ensure that all of the bits arrive simultaneously.

3. Type *easyplot* from within the Easytrax directory to create the gerber plots. A check plot can be printed from here. A gerber file must be created for the top and the bottom sides of the board (.gbl for the bottom, .gtl for the top), assuming a two-sided board. Also, the aperture file (standard.apt) should be printed out for later use by the gerber editor.

4. The gerber editor, the outline editor, and the milling machine are owned by the NPS Physics Department and are located in the basement of Halligan Hall (656-2065), adjacent to the linear accelerator. They have the ability to make multi-layered boards up to four layers thick. The Space Systems Academic Group also have similar equipment in Bullard Hall. This board was fabricated on the equipment owned by the Physics Department.

5. The gerber files should be viewed in the gerber editor prior to sending to the outline editor. Copy them to the PCGerber directory (c:\cam) and type *cam* to enter the editor. A design file and an aperture file must first be created. A prompt will appear asking for this info. By typing in a name that doesn't currently exist, the files will be created. The aperture file must contain identical apertures to those in the original design. The board layers must also be loaded prior to viewing.

5. Any design code errors that occur on loading are probably due to undefined apertures (D-codes). The aperture list can be modified under the *file* pull-down menu.

6. Modify the design as necessary. For the testbench, larger thermal relief pads were substituted for the power and ground

pads. This is simply a matter of substituting the D-code for the .062" thermal-relief pad for the D-code of the power or ground pad. This must be done on one side at a time or the thermal will be placed on both sides of the board. One side must be isolated by placing a standard .062" pad there and the other must have the thermal pad. Save the files and exit.

7. The gerber files must be edited to remove the *G01* code. It will show up only once per file and is placed automatically every time the file is saved by PCGerber. The DOS text editor and its search function can be used for this. This code instructs the milling machine to do a linear interpolation. The equipment owned by the Physics Department doesn't understand this command and will crash. Removing the code will prevent this.

8. The gerber files must be copied to the input directory of Protoboard (c:\pboard\input) and renamed as L1.GBR and L2.GBR (assuming only two sides).

9. Type *p1* to execute a batch file which starts Protoboard. Accept all the defaults unless you want to name the job, which isn't required. This program takes hours to run for any normal sized circuit.

10. If the Protoboard completes the design with no errors, the completed outline gerber files will be in the output directory (c:\pboard\output) and will be renamed *L1OUT.GBR* and *L2OUT.GBR*. There will also be two drill files (a .gbr file for viewing and a .dbf file for running on the milling machine) for each side which should be identical, and two files named *L1TXT.GBR* and *L2TXT.GBR*. These last two files are the same as the input files except that they have been reordered and cleaned up a bit.

11. Quite likely, the program will hang up in an endless loop. In this case, the working gerber file, *LOUT.GBR*, can be

exported to the gerber editor to find the error. It is located in Protoboard's working directory, c:\pboard\ex1. Copy it c:\cam and load it along with the corresponding gerber file for that particular side.

12. The error will be where the outline stops. There may be other errors that show up but that didn't hang up the program. These errors are usually not obvious but deleting the component that caused the error and replacing it will usually do the trick. Often, there are hidden segments of traces under the trace that only show up after the trace is deleted and the circuit redrawn. On the test bench, there were so many errors that it soon became apparent that redrawing the whole board would take less time. This was made relatively painless by displaying L1.GBR and L1TXT.GBR at the same time. They should both be identical. Make one a reference and delete all the traces in the other. In this fashion, the reference file shows the location of all the traces and the job becomes fairly simple. Redrawing all the traces on one side of the testbench took about two hours.

13. Start over at step 7 until Protoboard successfully completes the design and exits. It is still necessary to view the output files to double check that there are no errors. Sometimes an error will get by without stopping the program.

14. Once the outline is generated and all errors are corrected, the output files must be left in the Protoboard output directory so the milling machine drivers can find them.

15. The driver for the milling machine is executed by typing "p2". Prompts will appear at the lower right on the screen.

16. The design must be aligned to the machine so that it fits on the board. One way of doing this is to start with the drill file using only the masonite backing plates. Another way is to create a file containing only the outline but that takes another step. Once the location of the design on the

backing plate is determined, any offset required can be entered.

17. The blank PC board to be etched should be lightly sanded prior to mounting on the PB800 milling machine. It is important to use a clean backing plate under the PC board. If a drill were to encounter a preexisting hole, there is a good chance of breaking the bit.

18. It is probably best to run the drill file first since this was probably used initially for alignment purposes and the bit is still in place. All the holes are .031". The depth of the drill should be set to just penetrate through the PC board and barely into the masonite. This should be done in a corner of the actual board. Protoboard allows the tool head to be manually moved in increments of the step size that is entered by the operator. On the PB800 milling machine, there are two set screws on the front of the router motor. The top one secures the motor in the bracket and must always be kept tight. The lower one secures the bit guard. This one must be loosened slightly while drilling to let the bit plunge through the board. The bits must be installed so that they are flush with the tool guard. The pneumatic motor requires over 80 psi of dry air to operate. The safety shield around the bit is required so that the vacuum cleaner will remove all the dust.

19. After the board is drilled, change the bits to the milling bit. It must be mounted flush with the tool guard, as before, but this time the lower set screw must be tightened. The depth of the cut must be set so that the width is .010". An optical loupe with a measuring scale attached can be used to measure the traces. It would be wise to test this in all four corners to ensure the board is level. If it is not, it can be shimmed up a bit with a piece of paper. Lightly spray the board with teflon or silicon lubricant to prevent abrasive wearing of the plastic channel width limiter foot and mill the

top side.

20.   Prior to milling the bottom side, the gerber file (L2OUT.GBR) must be mirrored. Protoboard will easily do this, but it must be commanded to do so.

21.   Turn over the PC board, lightly sand this side, and remount. Lightly spray the board with silicon or teflon lubricant and etch the bottom side.

22.   After milling, remove the PC board and manually insert eyelets in each of the vias from the bottom of the board so that the heads are down. Remount the board with the top side upward. The eyelets fit fairly snug so there shouldn't be much problem with them falling out as the board is turned over.

23. Mount the swaging bit into the machine. Set the machine depth so that the suspension gap is just flexed when the eyelet is swaged. Set the RPM to one third of full speed by setting the air pressure to 20 psi. Spray the board with a silicon or teflon-based lubricant to keep the swaging bit from breaking. Run the drill file.

24.   The finished PC board can be cut from the remainder of the board with a sheer (the machine shop in the basement of Spanagal Hall has one), or Protoboard can do it. In order to have Protoboard do it, an outline of the board must be created. This only takes a few minutes. Any one of the layers can be made into an outline file simply by deleting all the pads, vias, and traces in the gerber editor. This must be run through the outline converter which only takes a few moments. A special bit is used to cut out the board and is mounted as the other types of bits. The speed of the cut is set to 5% and the depth is adjusted to just penetrate the board.

25.   Once the board is finished and and has functionally tested, it would be wise to protect it with conformal coating.

This will prevent dust from shorting a trace to the power or ground plane. It will also add a small measure of safety (maybe only to the fuses) by insulating the power and ground planes.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Suite 0944
   Ft. Belvior, VA 22060-6218

2. Dudley Knox Library, ... 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Commandant (G-SRF) 2
   United States Coast Guard
   Washington, DC
   20593-0001

4. Chairman, Code EC 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

5. Prof. Douglas Fouts, Code EC/Fs 2
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

6. Prof. Todd Weatherford, Code EC/Wt 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

7. LT John A. Thompson 1
   399D Ricketts Road
   Monterey, CA 93940